Trabajo Fin de Grado

Implementación de una solución CDN sobre OpenStack

Grado Ingeniería Telemática
(Plan 2008)

Tutor : Jaime Garcia Reinoso
Autor: Fernando Cerezal López

Resumen

La fuerte demanda de contenidos que existe actualmente en Internet ha hecho que aparezcan soluciones a medida que permitan a los grandes distribuidores de contenidos centrarse en su negocio y delegar la entrega todo aquel contenido que, siendo necesario, no aporta valor. Tales son las redes de entrega de contenidos, CDN por sus siglas en inglés.

Por otro lado, la fuerte variación de la demanda de actividad por parte de los usuarios ha hecho que surjan nuevas formas de gestionar los recursos de sistemas de información que ya no deben dimensionarse a su máximo de usuarios, si no crecer o disminuir disminuir para que el coste de prestar servicio sea proporcional a la cantidad de usuarios que lo demandan. De este modo han surgido las tecnologías denominadas en nube.

Este proyecto aúna ambas partes, montando una infraestructura de red de entrega de contenidos sobre una infraestructura de tipo nube. El proyecto parte de una infraestructura básica de tipo nube, montada en un trabajo de fin de grado anterior, y muestra como instalar y configurar los servicios de nube necesarios para prestar un servicio de almacenamiento de objetos y escalado de nodos bajo demanda, agrupado con un desarrollo a medida que explota estos servicios y balancea la carga en función del escalado.

Una vez finalizado, este proyecto proporciona un entorno de laboratorio para probar como es el funcionamiento de una red de entrega de contenidos y como una infraestructura de tipo nube es capaz de prestar los servicios necesarios para que la red de entrega de contenidos se centre únicamente en su cometido.

Índice

1. Tabla de contenidos

Resumen....2
Índice....3
Capítulo 1 - Introducción....5
Motivación....6
Encuadre socio-económico....6
Capítulo 2 – Estado del arte....7
Estado del arte....7
Capítulo 3 – Objetivo y planteamiento....12
Objetivo del proyecto....12
Entorno inicial....12
Planteamiento....13
Arquitectura....14
Arquitectura General....14
Balanceador DNS....19
Módulo de servicio CDN....20
Alternativas de diseño....22
Capítulo 4 – Implementación....24
Implementación....24
Selección de tecnologías....24
Desarrollo y Configuración....26
Capítulo 5 - Pruebas....77
Entorno de pruebas....77
Protocolo de pruebas de balanceador DNS....77
Protocolo de pruebas de servicio de CDN....78
Protocolo de pruebas notificador....79
Realización de pruebas de carga....79
Capítulo 6 - Planificación....81
Diagrama de Gantt....82
Capítulo 7 - Marco regulador....83
Capítulo 8 - Conclusiones....85
Trabajo futuro....86
Capítulo 9 - Presupuesto....87
Presupuesto....87
Referencias....89
Glosario....90
ANEXO I - Código de servicio CDN....92
ANEXO II - Script de arranque Notificador DNS....93
ANEXO III - Código balanceador DNS....94

ANEXO IV - Script de inicio del servicio CDN....98
ANEXO V - Script de arranque balanceador DNS....99

Indice de imágenes

Figura 1: Entorno inicial....14
Figura 2: Red inicial....15
Figura 3: Red final....15
Figura 4: Entorno final....16
Figura 5: Diagrama de red en nube....17
Figura 6: Esquema balanceador DNS....19
Figura 7: Esquema Aplicación CDN....20
Figura 8: Interfaz gráfica Jmeter....67
Figura 9: Configuración primer parámetro Jmeter....68
Figura 10: Configuración parámetros de prueba Jmeter....69
Figura 11: Adición de hilos concurrentes Jmeter....70
Figura 12: Configuración de hilos concurrentes Jmeter....70
Figura 13: Adición de bucle Jmeter....71
Figura 14: Configuración bucle Jmeter....72
Figura 15: Adición de muestreador Jmeter....73
Figura 16: Configuración muestreador Jmeter....73
Figura 17: Adición de muetreador retardante Jmeter....74
Figura 18: Configuración muestreador retardante....74
Figura 19: Bucle con todos los elementos Jmeter....75
Figura 20: Guardado plan de pruebas Jmeter....76
Figura 21: Diagrama de Gantt....82
Figura 22: Presupuesto....88

Capítulo 1 - Introducción

Introducción

Actualmente, la gran variabilidad de usuarios que utilizan cualquier plataforma disponible al público hace que sea necesario adecuar la capacidad de los sistemas a la demanda para satisfacerla sin incurrir en costes excesivos. Además, los contenidos deben poder llegar a los usuarios lo antes posible para ofrecer una mejor experiencia de usuario.

Con el objetivo de permitir que cada servicio se centre en su aporte de valor y evite necesitar medios para entregar a los usuarios todo aquello que rodea al servicio, como todo el contenido estático, o para poder evitar cualquier tipo de efecto indeseado por el tránsito de contenido por Internet, tales como jitter[1] o pérdidas de paquetes, se crearon las infraestructuras de redes de contenidos.
Una red de contenidos (CDN) se compone de una serie de nodos donde los clientes pueden almacenar contenidos y se les entrega algún tipo de identificador de ese contenido. Este contenido es automáticamente replicado entre los nodos de la red de contenidos. El identificador entregado al cliente es el que este utiliza para entregar ese contenido a sus usuarios. De este modo, cada vez que un usuario solicita este contenido, como puede ser una imagen o un vídeo, se lo solicita a la red de contenidos en lugar de al proveedor original, descargando a este de toda la infraestructura necesaria.
Para ofrecer una mejor calidad de servicio, las organizaciones con CDN introducen sus nodos en los puntos de intercambio de tráfico (Puntos neutros) o, directamente, en las infraestructuras de los proveedores de servicios de internet, esto es, los operadores de telecomunicaciones.

La misma variabilidad en la gestión de la demanda citada antes se modeló en Amazón, una de las mayores tiendas mundiales, mediante una plataforma que permitiese poner en producción una cantidad mayor o menor de sistemas dependiendo de la demanda de forma flexible, denominada Amazon Web Services[2]. Esta plataforma separó en servicios cada pieza de la producción, desde la electrónica a los servicios al público, haciendo que cada pieza pudiese solicitar a otras lo necesario para realizar su trabajo, perdiéndose la perspectiva de que un servicio se prestaba desde una infraestructura concreta. Este modelo es lo que terminó denominándose “nube”.

[1] El jitter es un efecto que aparece cuando los datos no llegan en el momento en el que deberían, causando que la señal tenga variaciones indeseadas como retardos o glitches. Un glitch, básicamente, es un salto abrupto en una señal que debería ser continua.
[2] https://es.wikipedia.org/wiki/Amazon_Web_Services

Una vez separados los servicios, empezaron a ofrecerse piezas separadas para cada servicio, apareciendo los términos Software as a Service, Infrastructure as a Service o Platform as a Service, dependiendo de si lo que se vendía al usuario era una capa de abstración a una solución, a una plataforma o a unos equipos.
Para la obtención de la flexibilidad que aporta una nube en una CDN, es posible dividir la arquitectura de la CDN en servicios y que cada uno de ellos se ejecute en un entorno de nube.

Motivación

El motivo para realizar este proyecto es poder utilizar una arquitectura CDN en un entorno docente o donde sean necesarias bajas prestaciones, de forma que esta arquitectura pueda ser replicada a escala y modificada bajo demanda.

Encuadre socio-económico

Desde el punto de vista docente, permitirá tener un simulador de proveedor de red de contenidos que pueda utilizar en un laboratorio de prácticas, lo que permitirá tener una experiencia más clara para los estudiantes sobre qué es y cómo funciona una red de contenidos.

Por otro lado, desde un punto de vista empresarial, permitirá a una pequeña o mediana empresa tener un entorno de pruebas donde comenzar a desplegar sus servicios de una forma parecida a como lo realizaría con un proveedor comercial de servicios en la nube.

Capítulo 2 – Estado del arte

Estado del arte

La cantidad de datos que son necesarios para ofrecer una experiencia adecuada a los usuarios crece continuamente. Esto ha llevado a los generadores de contenidos a buscar soluciones que les permitan minimizar sus costes de infraestructura. Un análisis de los datos contenidos en cualquier interfaz web permite ver que una gran cantidad de los datos se repiten en todas las pantallas de la interfaz, tales como logos, iconos, archivos css, estructura básica y demás. Toda esta información, que no varía, a la vez es la más solicitada por los usuarios. Por tanto, ofrece varias ventajas al generador de contenidos el hecho de delegar la entrega de esos datos. Por estas razones nacieron las redes de contenidos, en adelante CDN por sus siglas en inglés.

Para ilustrar cómo funciona una CDN se debe conocer cómo funciona un navegador web y cómo están estructuradas las páginas web. Una página web está compuesta por una serie de elementos definidos por etiquetas, utilizando el formato de marcada HTML, como la siguiente:

```
<html>
<head>Título de la página</head>
<body>
<img src=”imagenCabecera.png”/>
<p>Ejemplo de párrafo</p>
<img src=”logo.png”/>
<p>Otro ejemplo de párrafo</p>
<img src=”imagen3.png”/>
</body>
</html>
```

Las etiquetas img definen imágenes y su propiedad src define la dirección URL de donde obtener esa imagen. Dado que se encuentran en el mismo servidor que el resto de recursos, no es necesario definir una dirección completa. No obstante, es posible modificar esas etiquetas de la siguiente manera:

```
<html>
<head>Título de la página</head>
<body>
<img src=”http://zone1.cdn.org/cliente1/imagenCabecera.png”/>
<p>Ejemplo de párrafo</p>
```


Otro ejemplo de párrafo

</body>
</htm>

De este modo, las imágenes del logo y de la cabecera se obtendrán de unas máquinas distintas a la imagen3. Esto es, cuando el usuario cargue esa página en su navegador, este buscará la imagen3 en el mismo servidor que que el resto de la página, pero las imágenes de cabecera y logo las obtendrá del sistema zone1.cdn.org. Esto permite que la organización se centre en que sus sistemas ofrezcan adecuadamente la imagen3, que sí puede tener valor para ella, pero delegue completamente en la CDN la entrega del logo y la cabecera de su página, que no le aportan valor.

Una red de contenidos es una arquitectura en la que existen una serie de nodos distribuidos geográficamente en zonas orquestados por un sistema de control central.
A continuación denominaremos clientes de la red de contenidos a los generadores de contenidos, ya que son quienes se benefician de ella, y usuarios a los usuarios finales que obtienen el contenido, siendo transparente para ellos que los datos provengan de la red de contenidos o del generador de contenidos original.
Con esta nomenclatura, los clientes de la red de contenidos almacenan en ella toda aquella información que no varía rápidamente y que es frecuentemente demandada por los usuarios. En adelante a los datos que los clientes almacenan en la red de contenidos lo denominaremos objetos, que pueden ser imágenes, archivos de texto, archivos multimedia o cualquier otro tipo de archivo. Una vez el generador de contenidos ha almacenados en la red de contenidos todos los archivos pertinentes, utiliza direcciones de la red de contenidos con los identificadores de los objetos en el contenido entregado a los usuarios en lugar de entregar ellos directamente el contenido. De este modo, cuando un usuario carga en su navegador el contenido entregado, el navegador, de forma transparente para el usuario, solicita los objetos a la red de contenidos en lugar de al generador de contenidos, descargando a este de toda la infraestructura y ancho de banda necesarios para entregar esos datos.

El orquestador de la red de contenidos se encarga de diseminar entre los nodos distribuidos geográficamente toda la información que le entregan los clientes. También es posible que no exista un orquestador como tal, si no que todos los nodos sean capaces de acceder a todos los objetos, descargando en el sistema de almacenamiento la distribución de los objetos en sí. Es de destacar que que un grupo completo de sistemas pueden servicio bajo un único dominio, realizándose un balanceo mediante la resolución DNS entre los distintos nodos, siendo esto completamente transparente al usuario. Los nodos se distribuyen geográficamente

de modo que los contenidos estén lo más próximos posibles a los usuarios, actuando a modo de caché[3]. Para minimizar la distancia entre los usuarios y los contenidos, estos se ubican físicamente en los proveedores de servicios de internet o, si existen, en los puntos de intercambio de tráfico entre proveedores, esto es, puntos neutros[4], tales como Espanix[5] o Catnix[6] en España.

Proveedores actuales de CDN

Actualmente existen varias empresas que proporcionan CDN de manera comercial, tales como Akamai[7] o OVH[8]. Estas empresas tienen puntos de presencia en todo el mundo permitiendo a sus clientes ofrecer servicios al usuario que puedan competir con otras empresas que tienen CDN propia, como el caso de Netflix[9] o Youtube[10].

Arquitectura en nube

La arquitectura en nube permite una alta flexibilidad en el despliegue de servicios. Esto es especialmente interesante en el caso de una CDN, ya que la demanda de acceso de los usuarios pueden variar dramáticamente, dependiendo del huso horario o, por ejemplo, campañas comerciales. En este caso, en una infraestructura tradicional, la CDN debería estar dimensionada al alza, de forma que ningún usuario obtuviese un mal servicio, pero esto obligaría a que la infraestructura estuviese sobredimensionada la mayor parte del tiempo. La utilización de un infraestructura en nube también permite abstraerse de ciertos servicios, como el de almacenamiento, o bajo qué sistema se está prestando servicio, ya que todo el direccionamiento se modifica dinámicamente.
La arquitectura se basa en tres conceptos: modularidad, sistema de mensajería, API.

La modularidad se explota mediante la división del funcionamiento completo de la infraestructura en servicios más básicos. Esto es,puede existir un servicio de inicia y elimina máquinas virtuales, un servicio como modifica la redes virtuales entre estas, un servicio que proporciona formas de almacenamiento o un servicio que identifica y autoriza usuarios.

Cada uno de estos servicios ofrece a los demás una API mediante las que los demás módulos pueden hacer peticiones de servicio. Lo adecuado es que estas API

[3] Memoria temporal utilizada para almacenar la información con la que se está trabajando.
[4] Ubicación física donde los operadores intercambian tráfico.
[5] http://www.espanix.net/
[6] http://www.catnix.net/
[7] https://www.akamai.com
[8] http://www.ovh.es
[9] https://www.netflix.com
[10] http://www.youtube.com

sean atómicas, esto es, que las llamadas sean sin estado y una sola llamada contenga toda la información para realizar la acción completa.

Por último, es útil que todos los módulos tengan noción de lo que realizan los demás módulos, por lo que se habilita un sistema de notificaciones mediante un sistema de mensajería. De este modo los módulos pueden comunicarse cuando no desean que otros módulos realicen ninguna acción concreta.

Ventajas de una nube

La principal ventaja de un nube es esta modularidad. Dado que actualmente se ha alcanzado un punto en la tecnología de computación en la que no es posible escalar verticalmente, es decir, no es posible obtener procesadores más rápidos, el crecimiento horizontal se ha hecho una necesidad. Cada uno de los servicio descritos anteriormente puede ejecutarse en un sistema distinto y, lo que es más importante, más de una instancia de cada puede utilizarse a la vez. Esto hace posible que una infraestructura crezca de forma sencilla en base a las necesidades de una organización y de forma horizontal. No obstante, para que una solución escale en la misma medida, esta debe haber sido diseñada bajo esta premisa, de forma que varias instancias de ella puedan ejecutarse en paralelo para obtener un mejor rendimiento.

Sistema de mensajería

La solución hace uso del sistema de mensajería de la plataforma, al igual que los propios módulos de la plataforma openstack. Un sistema de mensajería se divide en un bus de mensajería, que recibe y entrega mensajes, y unos agente, que se conectan al bus, emitiendo o consumiendo mensajes. Dentro del bus de mensajería pueden existir canales, de forma que si un agente envía un mensaje a un canal todos los agentes conectados al canal lo reciben. De este modo el remitente no necesita conocer a los destinatarios. El bus utilizado en este caso es RabbitMQ, que denomina a los canal es de comunicación “virtual host”. Este tipo de comunicación es utilizado por los módulos de openstack para notificarse eventos entre sí y en el balanceador DNS para comunicarse entre los nodos de la CDN y el servidor DNS. El protocolo que utiliza este sistema de mensajería se denomina AMQP.

Otros soluciones similares

Al realizar una búsqueda sobre CDN realizadas sobre openstack, sólo se han encontrado dos proyectos. Por un lado existe el proyecto Poppy[11], que una vez analizado en realidad es un interfaz de openstack a CDN comerciales. Por otro lado está el proyecto de pkgclod en githib[12], que adolece de falta de mantenimiento y documentación. Por tanto, no se ha encontrado un proyecto similar a este actualmente.

NOTA: A lo largo de esta memoria se utilizarán el símbolo $ cuando algo pueda ejecutarse como usuario normal y el símbolo # cuando algo deja ejecutarse como usuario administrador.

[11] https://wiki.openstack.org/wiki/Poppy
[12] https://github.com/pkgcloud/pkgcloud/blob/master/docs/providers/openstack/cdn.md

Capítulo 3 – Objetivo y planteamiento

Objetivo del proyecto

El objetivo del proyecto es implementar un entorno CDN, esto es, que permita entregar objetos mediante peticiones HTTP, utilizando un entorno de nube para incrementar los nodos de servicio bajo demanda y utilizando un balanceo mediante DNS para repartir las peticiones entre los nodos disponibles en ese momento.
Dado que es un entorno de laboratorio no se tienen en cuenta aspectos de seguridad y se busca la mayor flexibilidad.

Entorno inicial

Toda la solución estará basada en el entorno implementado por Jesús Sánchez Manzanero para su Trabajo Fin de Grado[13], por lo que se respetará su arquitectura, convenciones y nombres asignados.

Dicho entorno está basado en OpenStack, en su versión Juno, que consta de nueve módulos:

- ❑ Keystone: Módulo de autenticación, autorización y descubrimiento de servicios
- ❑ Nova: Módulo de computación. Inicia y termina instancias virtuales
- ❑ Neutron: Módulo de gestión de redes virtuales
- ❑ Cinder: Módulo servidor de dispositivos de bloques, tales como particiones o volúmenes lógicos.
- ❑ Glance: Servidor de imágenes para máquinas virtuales
- ❑ Horizon: Módulo de cuadro de mando y control mediante interfaz web
- ❑ Swift: Módulo de almacenamiento de objetos.
- ❑ Ceilometer: Módulo de monitorización.
- ❑ Heat: Módulo de orquestación, recogiendo información de monitorización y realizando acciones en consecuencia.

No obstante, el proyecto realizado por Jesús Sánchez sólo implementa los módulos: Keystone, Nova, Neutron, Glance, Cinder y Horizon. Para realizar este proyecto es necesario utilizar, además de estos, los módulos Swift, Ceilometer y Heat, por lo que se configurarán en este proyecto.
Por tanto, toda la configuración de los módulos instalados por Jesús Sánchez se omitirá y se centrará en el montaje de un servicio de red de entrega de contenidos y de los módulos necesarios. Por otro lado, el entorno montado no estaba preparado

[13] Diseño e Implementación de un laboratorio de Openstack - Jesús Sánchez Manzanero - 2015

para montar servicios añadidos sobre él, por lo que es necesario hacer uso de tecnologías auxiliares que se describirán en cada apartado.

Planteamiento

El planteamiento inicial parte de la idea de ofrecer los servicios de una CDN basándose en los servicios de una infraestructura de tipo de nube.

A lo largo de este proyecto utilizaremos el concepto de zona como ubicación a la que se presta servicio. Una zona presta servicio bajo un dominio mediante uno o más nodos.
Se denominará cliente a cada ente que almacene objetos en la CDN y usuario a quien los solicita mediante los nodos de servicio de la CDN.

Para ello, la solución constará de tres partes a desarrollar, además de la instalación de los módulos de openstack necesarios:

Balanceador DNS:

> Este componente se basa un servidor dns cuyos registros cambian dinámicamente en función de los nodos disponibles. Es caso de existir más de un nodo para servir una zona, el servidor DNS realiza una iteración round robin sobre ellos en la resolución del dominio.

Servicio de entrega de contenidos

> Este servicio se ejecuta en cada nodo del gestor de contenidos. A él llegan peticiones HTTP de los recursos que el cliente haya registrado en la CDN, el componente obtiene el recurso de la plataforma y lo entrega.

Generación de instancias bajo demanda

> Esta parte de la solución será realizada por los módulos de openstack necesarios, haciendo que haya más o menos nodos en funcionamiento, y registrado en el balanceador DNS en función de la carga que estos tengan.

Los objetos se almacenan dentro de contenedores. En este proyecto utilizaremos los contenedores como espacios de nombres, aunque sea posible configurar muchos otros parámetros sobre el almacenamiento. De este modo, utilizaremos un contenedor por cada cliente, así conseguiremos que los nombres de los recursos sean únicos dentro de cada cliente, pero puedan repetirse entre clientes.

Por cuestiones de sencillez, tanto el balanceador como el servicio de entrega de contenidos se instalará en la misma instancia para utilizarla como imagen base de las instancia, utilizándose el primer nodo como DNS y CDN y las demás instancias únicamente como CDN.

Arquitectura

Arquitectura General

En el siguiente diagrama se muestra el entorno inicial:

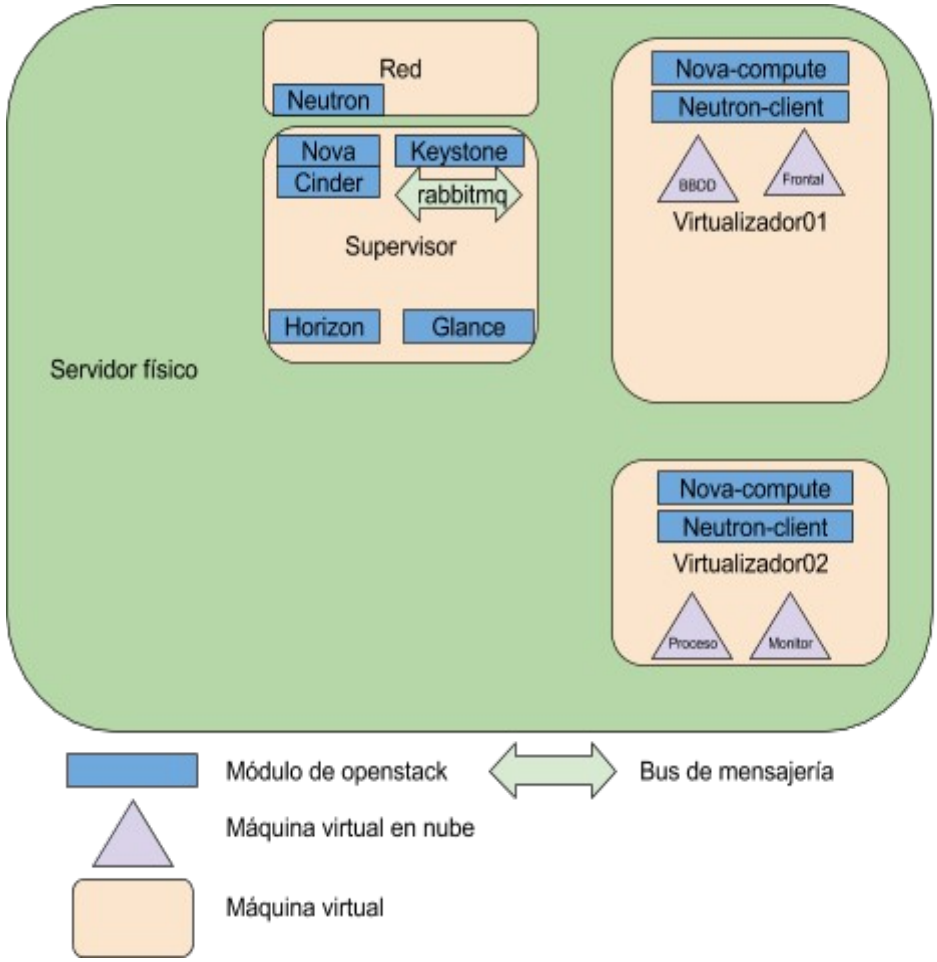

Figura 1: Entorno inicial

En este esque se pueden apreciar todos los módulos instalados inicialmente.
En este entorno existe un sistema supervisor que contiene todos los servicios de openstack. Los sistemas virtualizador01 y virtualizador02 contienen los agentes de los módulos nova y neutron que permiten levantar instancias en ellos y dotarlos de comunicaciones. También existe un servidor red que actúa a modo de router y orquesta a sus agente en los virtualizadores.

Este entorno permite instanciar máquinas virtuales, pero no permite almacenar objetos en la infraestructura de openstack ni permite monitorizar las instancias. Tampoco tiene instalado el orquestador de openstack, heat, por lo que no puede dotar de inteligencia a la nube que se adecúe a la demanda.

También es necesario tener en cuenta el esquema de redes ya existente en el entorno. El siguiente esquema forma parte del Trabajo Fin de Grado de Jesús Sanchez Manzanero:

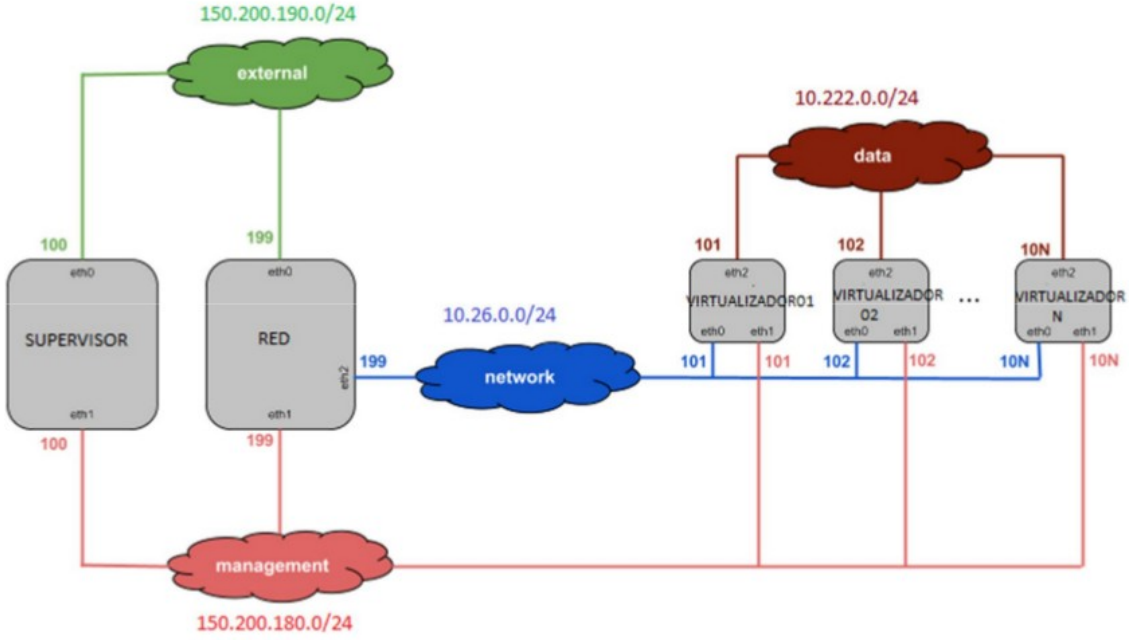

Figura 2: Red inicial

En este esquema se pueden ver las diferentes redes configuradas virtualmente en el entorno.
El esquema final respecto de la red será el siguiente:

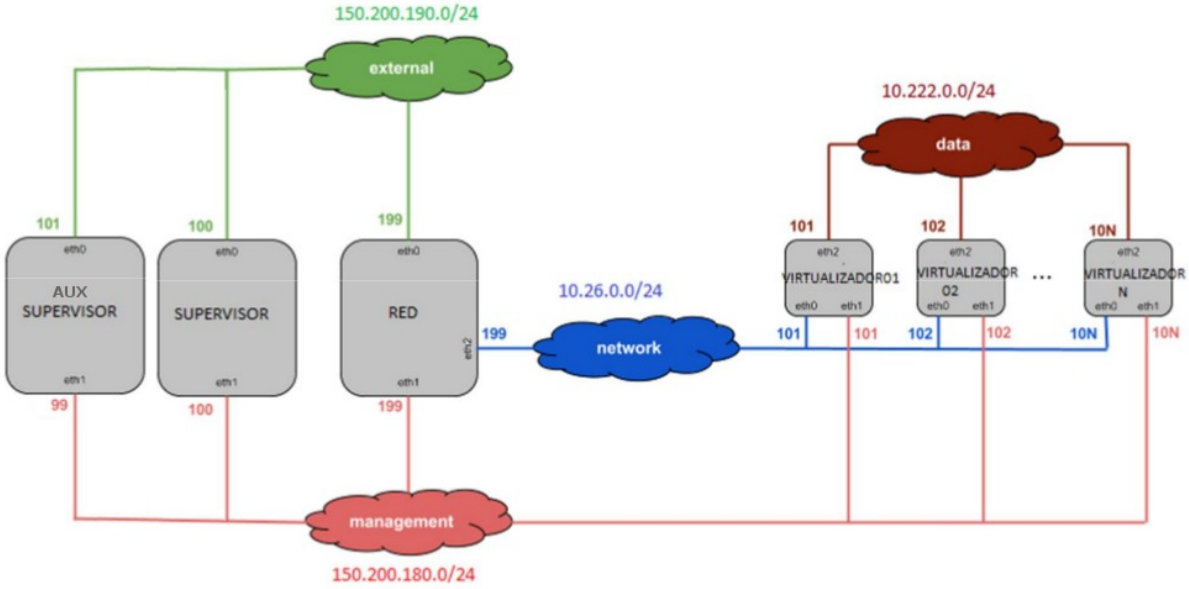

Figura 3: Red final

Y respecto de los sistemas y módulos se llegará a la siguiente situación:

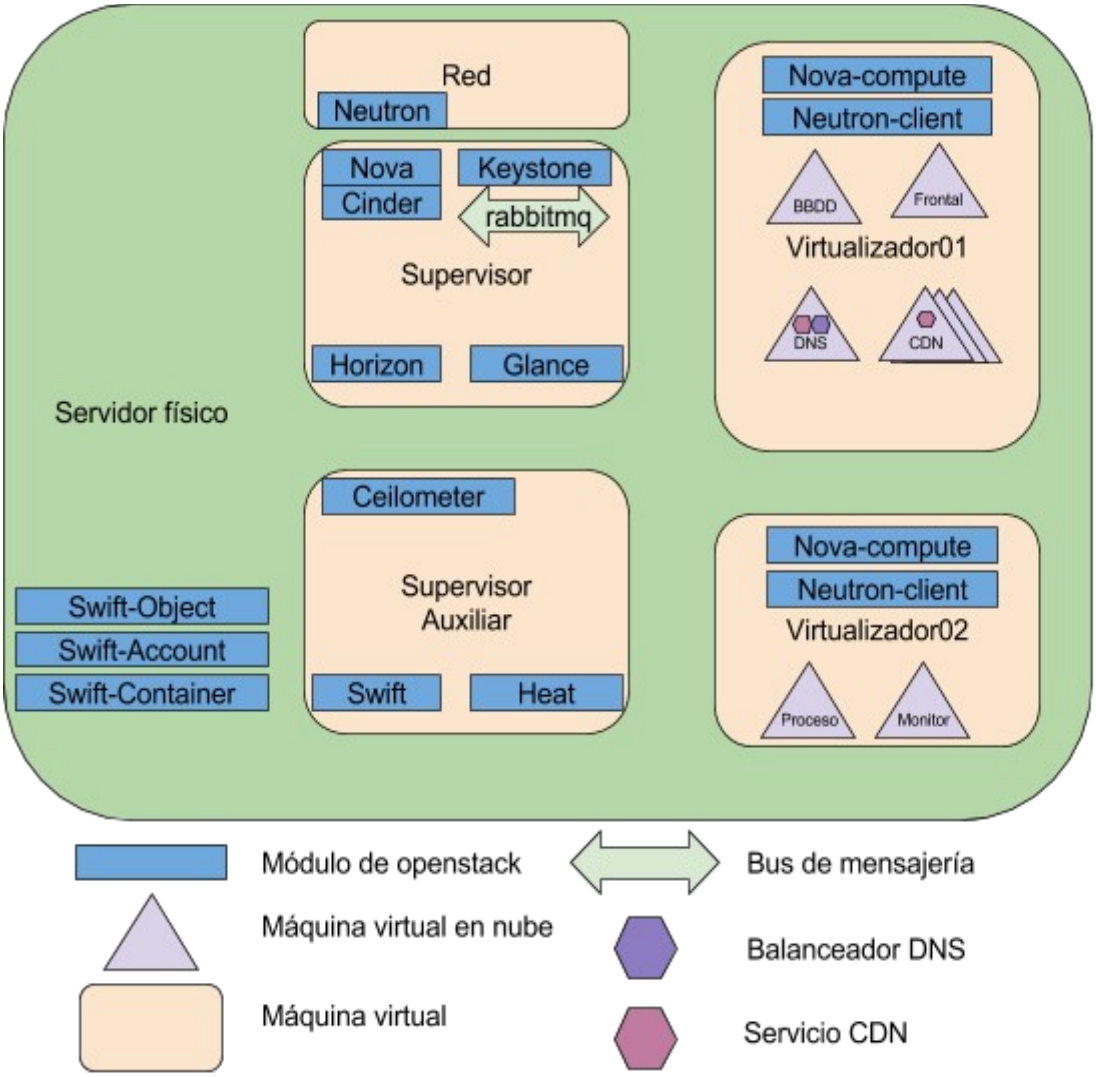

Figura 4: Entorno final

En él se puede apreciar que los sistemas de la CDN y del servicio DNS se ejecutan en el virtualizador01, que ha sido necesario adecuar para poder alojarlos. También se observa un nuevo servidor supervisor auxiliar que ha sido necesario instalar, ya que el servidor supervisor no tenía recursos suficientes para alojar todos los servicios. En él se han alojado los tres servicios de openstack que no existían en el entorno anterior.

En cuanto al entorno en nube, sus comunicaciones se muestran en el siguiente diagrama:

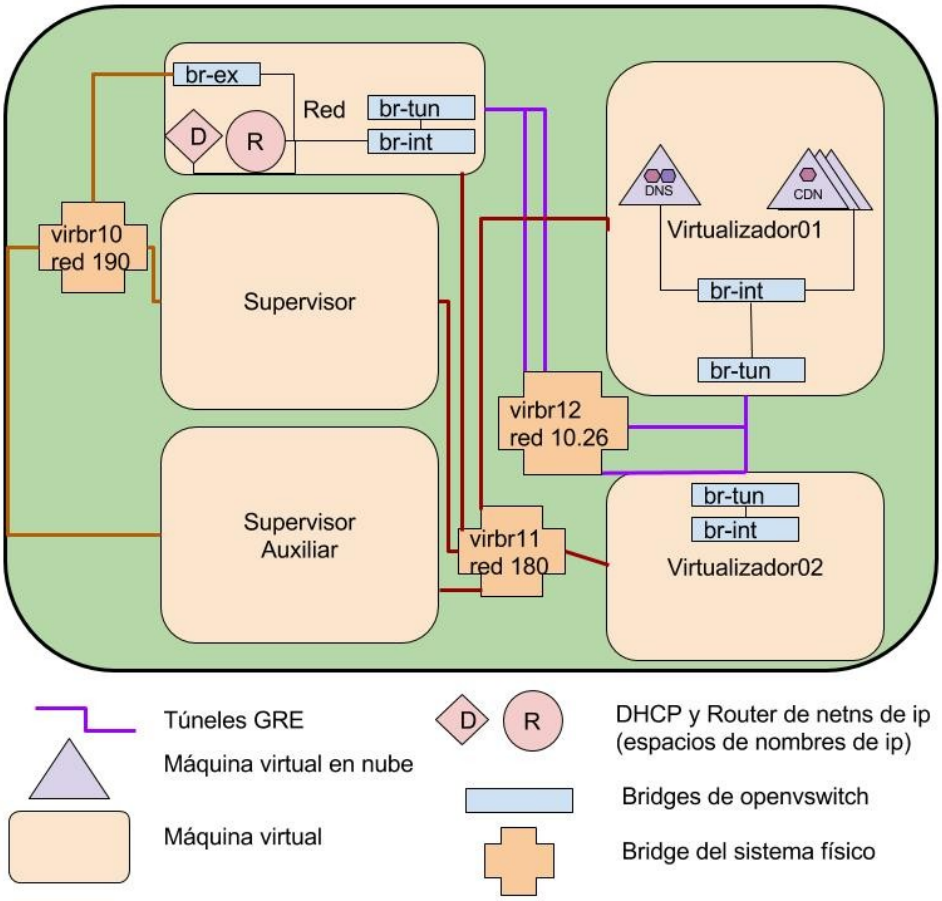

Figura 5: Diagrama de red en nube

Por otro lado, es necesario aclarar que los agentes de swift encargados de almacenar realmente los objetos están servidos desde el servidor físico, ya que este tenía asignado todo el espacio libre del entorno. También ha necesario instalar un servidor de nfs en el servidor físico que que ofreciese almacenamiento al servidor supervisor, ya que no existía espacio suficiente para que el servicio Glance pudiese almacenar imágenes de máquinas virtuales más allá de centenares de megabytes.

Por último, ha sido necesario modificar el almacenamiento, la cantidad de memoria RAM y de procesadores en el virtualizador01, ya que no contaba con recursos suficientes y, aunque tenía unos bloques de almacenamiento asignados, estos no estaban adecuados para su utilización.

Para llegar a este entorno se han desarrollado cinco fases:

- ❑ Balanceador DNS: los nodos con el servicio CDN notifican a este balanceador su existencia, de forma que este itera la resolución del nombre de su zona sobre ellos.

- ❑ Módulo de servicio CDN: Recibe las peticiones de los usuarios para objeto, obteniendo este objeto a través de la biblioteca Swift de OpenStack y se lo entrega a los usuarios

- ❑ Creación de imagen de nodo de servicio y supervisor auxiliar, instalando los módulos swift, ceilometer y heat.

- ❑ Adecuación del entorno: Modificación de los sistemas de infraestructura existente para poder alojar estos servicios.

- ❑ Configuración de servicio, orquestación y pruebas

Balanceador DNS

La arquitectura del balanceador DNS se basa en dos componentes que se comunican mediante el sistema de mensajería de la infraestructura del siguiente modo:

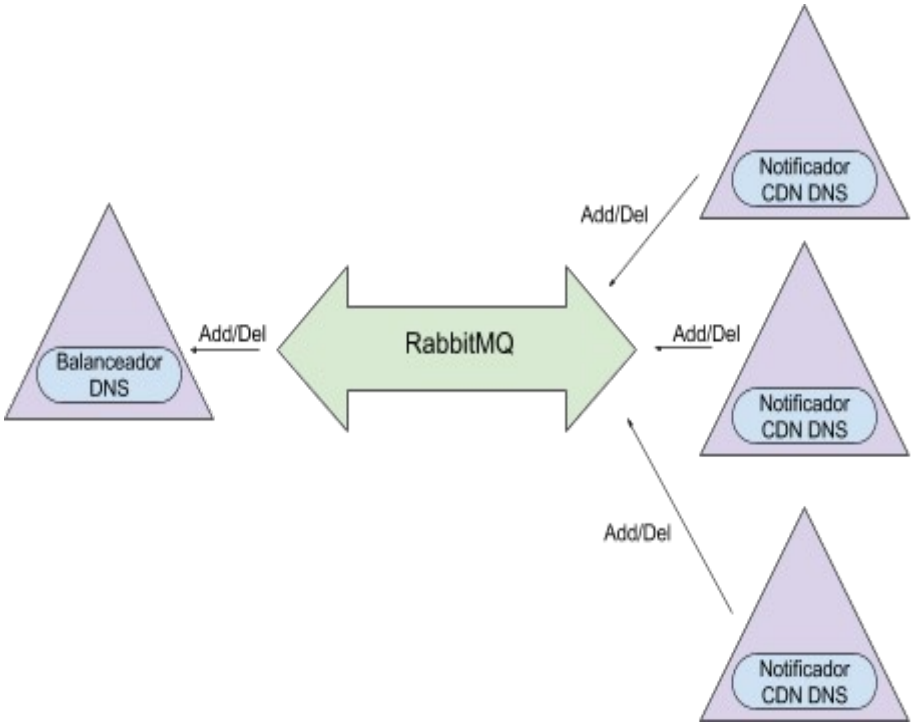

Figura 6: Esquema balanceador DNS

Los distintos nodos de servicio ejecutan un programa en su arranque que notifica al balanceador DNS su existencia para dar servicio, mediante la orden add y los parámetros correspondientes, y notifican de nuevo que dejan de prestar servicio al apagarse el sistema, mediante la orden del. El balanceador recoge estas peticiones y registra o elimina las direcciones de los nodos para el dominio correspondiente. El balanceador realiza un balanceo de tipo round robin sobre todos los nodos que resuelven para un mismo subdominio. Por simplicidad, el dominio que se resuelve está configurado en el programa del balanceador, estando establecido al dominio cdn.org.

Módulo de servicio CDN

El módulo de servicio de la CDN es un pequeña aplicación web que permite que se soliciten recursos mediante el protocolo HTTP, obteniendo los recursos de la plataforma openstack mediante el uso del módulo swift.

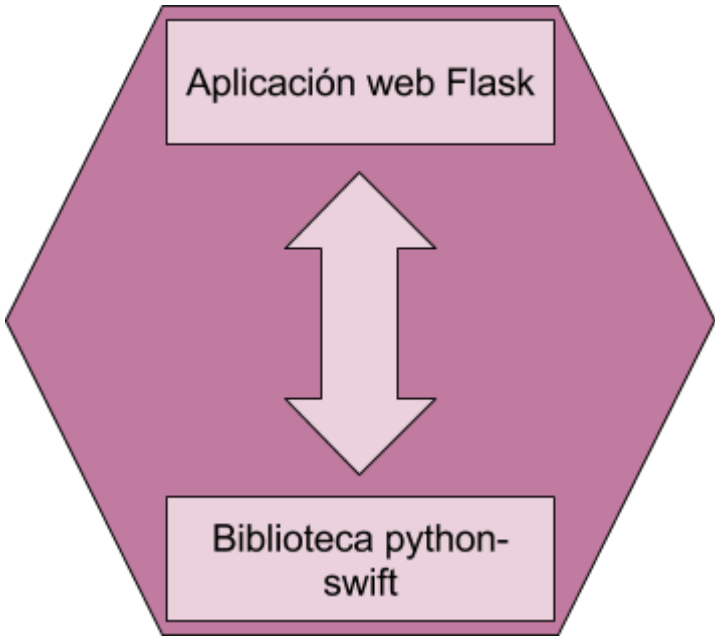

Figura 7: Esquema Aplicación CDN

De este modo se ponen los recursos almacenados en la CDN disponibles mediante acceso HTTP. Para adecuarlos a los requerimientos, la aplicación web recibe peticiones a dirección del tipo http://zone1.cdn.org/cliente1/logo.png, donde logo.png es el recurso a obtener, cliente1 es el cliente que ha almacenado el recursos, y que se hace coincidir con un contenedor de swift, utilizando espacios de nombres distintos para cada cliente.

Creación de nodo de servicio y supervisor auxiliar

Dado que la infraestructura existente no permitía instalar más servicios, se ha implementado un supervisor auxiliar donde se ha instalado los módulos swift, heat y ceilometer. En esta fase también se ha creado una imagen donde está instalado tanto el software de servicio de CDN como el balanceador DNS para que actúe de imagen base al levantarse las instancias.

Adecuación del entorno

Han sido necesarias una serie de acciones para modificar las capacidades de los sistemas existentes de forma que pueda prestar los servicios requeridos. Estas acciones son la modificación del almacenamiento, memoria RAM y procesadores virtuales del servidor virtualizador01 y la instalación de un servidor de NFS para que el servidor supervisor pudiese acceder a más almacenamiento.

Configuración de servicio, orquestador y pruebas

El orquestador de openstack está implementado en el módulo Heat. Este módulo permite definir plantillas que definen comportamientos. El módulo Heat se informa del estado de la infraestructura mediante el módulo ceilometer, cuyo cometido es la monitorización. Mediante las plantillas se definen unos umbrales para los parámetros que puede proporcionar ceilometer y, del mismo modo, se definen unas acciones a ejecutar en caso de traspasar dichos umbrales. En ese caso, Heat ordena a los módulos correspondientes realizar las acciones necesarios. En este caso, la monitorización se realizará en base a la carga de los nodos y la acción a tomar será provisionar más nodos. También es necesario configurar los parámetros que tendrán los nodos de servicio y realizar pruebas de carga.

Alternativas de diseño

- **Imacenamiento de objetos mediante otra tecnología**
  Sería posible almacenar objetos mediante otra tecnología, por ejemplo almacenando los objetos directamente en un árbol de directorios adecuado y ofreciéndolo mediante un protocolo por red (NFS, SAMBA) a cada nodo de servicio. De este modo no sería necesario utilizar swift. No obstante, en ese caso el almacenamiento sería parte de la solución CDN y, por tanto, responsable de su replicación a otras zonas. También sería necesario que el servidor que contuviese los datos estuviese en una ubicación conocida por los nodos antes de su instanciación, lo que no es necesario con swift, al ser parte de la infraestructura.
- **Monitorización externa y sin orquestador**
  Sería posible monitorizar la carga de los nodos mediante otra tecnología, como Nagios u otro sistema de monitorización. Al ser la instanciación de un nuevo nodo simplemente una llamada a la API HTTP de nova, es posible configurar esta llamada en el sistema de monitorización como acción en caso de superar un umbral.
- **Utilización servidor web como servicio en lugar del integrado en flask**
  El microframework flask contiene un servidor integrado que es el utilizado para prestar servicio. Este servidor se inicia al ejecutar la aplicación hecha en python. Sería posible utilizar otro servidor, como Apache con su módulo de python, para ejecutar este servicio. No obstante, la adición de esta complejidad a la plataforma no aporta ningún valor.
- **Comunicación entre nodos y servidor DNS**
  Inicialmente la forma de comunicación entre el servidor DNS y los nodos se pensó haciendo que los nodos de servicio creasen un archivo vacío cuyo nombre contuviese su zona y dirección IP en un directorio del servidor DNS utilizando SSH. Sin embargo, esto obligaba a configurar el intercambio de claves entre nodos y servidor DNS y a que los nodos conociesen de antemano la dirección del servidor DNS, ya que no existe otro servidor DNS en el sistema. Utilizando el sistema de mensajería se eliminó toda necesidad de configuración en los nodos, dejando sólo la configuración de conexión al bus de mensajería.
- **Modificación de servidores para adecuarlos a los nuevos requisitos**
  Sería posible modificar todos los servidores existentes y adecuarlos para las nuevas necesidades de la plataforma completamente, eliminando todos los servicios del servidor físico, ya que no es el sitio indicado para configurarlos. Sin embargo, el hecho de respetar que continúe funcionando la configuración anterior hace que esto no sea posible.

Puntos de mejora

Durante el desarrollo del proyecto se han encontrado diversos problemas que se han solventado convenientemente, pero que sería más adecuado dedicar un mayor esfuerzo a mejorar:

- **El servidor dns necesita ser más rápido**
  Se ha utilizado una biblioteca en python para realizar el balanceador DNS debido a que era sencillo conectarlo con el bus de datos de la plataforma y era rápido su desarrollo. No obstante, ante una tasa alta de peticiones ha demostrado inestabilidad. Sería conveniente rehacerlo con otra tecnología.
- **El servidor de imágenes debe tener espacio propio para almacenar las imágenes**
  Debido a que la plataforma inicialmente estaba pensada para sistemas mucho más pequeños, el servidor supervisor no tenía espacio suficiente para almacenar imágenes de gran tamaño, ya que todo el espacio estaba asignado al servidor físico. Para solventarlo, se habilitó un servidor de NFS en el servidor físico cuyo montaje está documentado en su apartado correspondiente. No obstante, sería más conveniente que fuese el espacio lo tuviese el sistema que proporcione el servicio, bien moviendo el espacio o bien moviendo el servicio.
- **El servidor de objetos y el de imágenes deberían tener particiones para el almacenamiento**
  El submódulo del servidor de objetos que almacena realmente los datos está habilitado en el servidor físico, al igual que se ha hecho para almacenar imágenes. Sin embargo, al estar en el mismo sistema de archivos, en caso de almacenarse demasiados objetos o imágenes se llenaría el disco raíz del sistema y este empezaría a fallar. Por ello, sería conveniente que los puntos de almacenamiento estén en sistemas de archivos distintos que, en caso de llenarse, no provocase un mal funcionamiento del propio sistema.
- **Debería existir más de un servidor de objetos**
  Parte de la versatilidad del almacenamiento de objetos en nube se basa en una abstracción total del almacenamiento y en que los objetos se replican desatendida entre los diferentes nodos de almacenamiento. Dado el entorno de trabajo, donde sólo hay un sitio para almacenar objetos, se ha implementado la solución con un sólo de almacenamiento y, por tanto, sin replicación. Sería conveniente que existiese más de un punto de almacenamiento y se hiciese uso de la replicación automática. En el caso de una CDN permite que el almacenamiento distribuido, parte básica del servicio, se realice de forma automática.

Capítulo 4 – Implementación

Implementación

Selección de tecnologías

A continuación se detallan los requisitos a cumplir y las tecnologías seleccionadas para cubrir la funcionalidad.

Balanceador DNS

Se busca una tecnología que permita:

- ❑ Recibir y entregar peticiones y respuestas mediante el protocolo DNS
- ❑ Modificar dinámicamente la base de datos de nodos activos
- ❑ Comunicación sencilla y fiable entre todo los nodos y el servidor DNS

Una tecnología que permite todo esto es:

- ❑ Lenguaje Python
- ❑ Biblioteca dnslib, que permite actuar como servidor DNS
- ❑ Biblioteca Pika que permite comunicarse mediante el bus rabbitmq ya existente en el sistema

Servicio de Entrega de Contenidos

Se busca una tecnología que permita:

- ❑ Ofrecer servicios vía web
- ❑ Ofrecer los propios objetos vía web
- ❑ Almacenamiento de objetos

Una tecnología que permite todo esto es:

- ❑ Lenguaje Python
- ❑ Microframework flask para aplicación web
- ❑ Módulo swift de opentack para almacenar objetos
- ❑ Biblioteca de integración con swift de python

Creación de imagen de nodo de servicio y supervisor auxiliar

Los servicio desarrollados se instalan en un sistema cuya imagen servirá de base para la instanciación de nodos.
Los requisitos son:

- ❑ Disponibles automáticamente las bibliotecas requisitos de los desarrollos
- ❑ Disponibles automáticamente los módulos de openstack en la misma versión ya instalada en la plataforma.
- ❑ Montaje sencillo de servicios
- ❑ Bajos requisitos de funcionamiento del sistema base

En base a estos requisitos se ha seleccionado la distribución Debian de GNU/Linux en su versión “stable”.

Adecuación de los servicios bajo demanda

Era un requisito del proyecto la utilización de la plataforma openstack ya existente, por lo que se utilizarán los módulos:

- ❑ Ceilometer para monitorizar la carga de las máquinas virtuales
- ❑ Heat para recibir datos de ceilometer y ordenar a nova lanzar o parar instancias nuevas de ser necesario.
- ❑ Nova para iniciar o parar instancias de nodos.

Adecuación del entorno

Ha sido necesario adecuar el entorno del siguiente modo:

- ❑ Modificación de virtualizador01, estando previamente configurado con virt-manager
- ❑ Modificación de supervisor, haciendo que tenga más espacio de almacenamiento para imágenes en el servicio glance

Para estas acciones se han seleccionado las siguientes tecnologías:

- ❑ Modificación de configuración utilizando la herramienta virsh de virt-manager
- ❑ Uso de servidor NFS para ofrecer almacenamiento del servidor físico al servidor supervisor

Pruebas

Para poder realizar pruebas de carga es necesario una herramienta que permita definir:

- ❑ Pruebas HTTP
- ❑ Retardos entre peticiones
- ❑ Distintos usuarios concurrentes

Una herramienta que cumple estos requisitos en la herramienta Jmeter[14] de Apache.

[14] http://jmeter.apache.org/

Desarrollo y Configuración

Balanceador DNS

El balanceador DNS consta de dos módulos:

- Notificador: El notificador es una pieza de software que notifica al Servidor DNS cada vez que un nodo nuevo se activa. Este componente se ejecuta en cada nodo de servicio, indicando el alta en el arranque de este y la baja cuando se el nodo se apaga. La ejecución de este módulo se realiza como un servicio del sistema. La notificación se realiza mediante un mensaje AMQP utilizando el servidor RabbitMQ de la plataforma.

- Servidor:El servidor DNS se comporta como un servidor DNS estándar en su interfaz hacia el usuario, recibiendo peticiones en el puerto 53. Internamente el servidor escucha mensajes AMQP y registrar o elimina los nodos notificados en las zonas correspondientes, así como registra o elimina zonas en caso de que no existiesen nodos anteriormente o, tras una eliminación, una zona no cuente con más nodos.

Dependencias

El servidor DNS se basa en dos bibliotecas para su funcionamiento:

- libdns: la biblioteca libdns permite el tratamiento de peticiones DNS para interpretar los parámetros y generar respuestas acordes el protocolo DNS según los parámetros dados.
- pika: La biblioteca pika permite conectarse a servidores AMQP desde programas escritos en python.

Configuración del sistema de mensajería

La comunicación entre el notificador y el servidor se realiza utilizando el servidor RabbitMQ de la plataforma. Para realizar la comunicación autenticada es necesario registrar a un usuario en el servidor de mensajería. En este caso se registra al usuario cdndns, con contraseña dndns, mediante la ejecución de la siguiente línea en el servidor donde este reside, esto es, en el servidor supervisor en nuestro caso:

```
# rabbitmqctl add_user cdndns cdndns
```

Conexión de un canal

La comunicación entre el notificador y el servidor se realiza mediante el canal "cdndns". Este canal se crea automáticamente en el servidor AMQP cuando el notificador o el servidor DNS se conectan a él.

Formato de mensaje

El mensaje enviando del notificador al servidor DNS tiene el siguiente formato:

“orden zona direcciónIP”

Donde orden puede ser:

1. add: para registrar un nodo nuevo
2. del: para eliminar un nodo

Zona es el subdominio a resolver. Esto es, si el dominio es cdn.org. y el nodo debe servidor bajo el dominio zona1.cdn.org., entonces el parámetro zona deberá ser "zona1".

DirecciónIP: Dirección IP del nodo que presta el servicio.

De este modo, el nodo que sirve en la zona1.cdn.org y con dirección IP 222.222.222.222 enviará el mensaje:
add zona1 222.222.222.222

cuando se inicie y el mensaje
del zona1 222.222.222.222

cuando se apague.

El servidor DNS tiene un registro en memoria, iniciándose sin nodos, en el que registran zonas nuevas cada vez que un nodo se inicie para una zona nueva, eliminan zonas cada vez que una de estas no tenga nodos que le presten servicio y realiza un balanceo de carga round robin en la resolución para todos los nodos que presten servicio para una zona.

El código del notificador se encuentra en el Anexo II y el código del balanceador DNS se encuentra en el Anexo III.

El notificador debe comunicar la dirección ip a la que responde del exterior, por lo que ha sido necesario desarrollar un pequeño ejecutable que obtenga su dirección externa para comunicarla por el notificador. Este pequeño ejecutable se denomina getexternalIP y se instala como utilidad del sistema. Su código se encuentra en el anexo VI.

Servicio de entrega de contenidos

El servicio de entrega de contenidos se basa en una pequeña aplicación web que es capaz de comunicarse con el servicio swift de openstack. De este modo, la aplicación web hace de interfaz al exterior de la plataforma.

Para hacer sencilla esta implementación se seleccionó el microframework flask que permite realizar pequeñas aplicaciones web de una forma rápida e intuitiva. Dado que las peticiones a la CDN se realizarán directamente por el navegador web del usuario y no existirá interfaz de usuario, se elige utilizar la propia URL de los objetas como de identificarlos en el espacio de nombres.

Diseño

El diseño se basa en hacer encajar la forma de interactuar con swift al obtener objetos con la forma en que se pueden hacer peticiones HTTP a una aplicación web.

La utilización de swift para almacenar objetos establece un marco de trabajo, ya que los objetos deben ser almacenados en contenedores. Una opción de diseño es utilizar un único contenedor para todos los objetos, almacenando los objetos con un identificador único y guardando en una base de datos intermedia la relación entre el nombre real y el identificador único. No obstante, con el objetivo de obtener las máximas prestaciones y simplificar la solución, se utilizan los contenedores de swift como espacio de nombres por cliente y se respetan los nombres de los archivos, de forma que la obtención de los recursos a partir de la petición es directa.

De este modo, se utilizan las rutas que se pueden usar en una petición web para distinguir los objetos de la siguiente forma:

http://zona1.cdn.org/cliente/recurso

Esto es, para el cliente redsocial y el objeto logo.png:

http://zona1.cdn.org/redsocial/logo.png

Es decir, el término redsocial es un subespacio dentro de la URL pero es a su vez el nombre del contenedor en swift. Del mismo modo, el archivo logo.png es el recurso que se solicita mediante HTTP y a su vez el nombre del objeto a solicitar a swift.

Internamente, la aplicación web solicitará el recurso logo.png en el contenedor redsocial y devolverá su contenido al usuario mediante la llamada a swift.

Requisitos

La aplicación web se basa en el microframework flask de python. Este framework permite abstraerse de la lógica de un servidor web y asignar funciones a rutas de dominio.

Esta aplicación también utiliza la biblioteca swift de python para obtener los recursos de la plataforma realmente.

Código

El código de este módulo se encuentra en el Anexo I.

La aplicación recibe a través de la llamada HTTP los parámetros antes mencionados e invoca al cliente para obtener el objeto. Una vez obtenido lo entrega mediante HTTP o devuekve un error 404 si no exsiste.

Creación de imagen de nodo de servicio

Para que todo lo anterior pueda prestar el servicio esperado, es necesario modificar el entorno en el que se basa el proyecto.

Inicialmente, se instala una máquina virtual nueva sobre una distribución que tenga empaquetados los módulos de openstack en su versión Juno. En este caso se eligió Debian GNU/Linux en su versión estable.

Para ello, creamos un nuevo disco virtual mediante la orden:

```
$ qemu-img create -f qcow2  cdn.qcow2 6G
```

que indica que será una imagen de tipo qcow2, que es la forma estándar, tendrá un tamaño de 6G y su nombre será cdn.qcow2.

Sobre él instalaremos una distribución debian estable mediante la instalación por red, que se puede obtener de la siguiente página:

```
https://www.debian.org/CD/netinst/
```

mediante la ejecución de la orden:

```
$ wget http://cdimage.debian.org/debian-cd/8.3.0/amd64/iso-cd/debian-8.3.0-amd64-netinst.iso
```

Una vez obtenida esa imagen, iniciaremos la máquina virtual mediante la ejecución de la orden:

```
$ kvm -m 1024 -hda cdn.qcow2 -cdrom debian-8.2.0-amd64-netinst.iso -boot d -netdev user,id=user.0 -device e1000,netdev=user.0
```

Esto indica que será una máquina virtual con 1024MB de ram, cuyo disco duro será cdn.qcow2, que tendrá la imagen descargada en su lector de cdrom virtual, que arrancará desde el cdrom y como será el acceso a la red.

Al ejecutar esto se abrirá una ventana en la que seguiremos el manual de instalación de Debian[15]. Tan solo tener en cuenta que sólo es necesario instalar el sistema base y el servidor de SSH llegado el momento.

[15] https://www.debian.org/releases/stable/amd64/index.html.es

En este sistema se registran el usuario root con contraseña root y el usuario user con contraseña user de cara a que se sencillo acceder y modificar por posteriores usuarios.
Una vez terminada la instalación, se apaga el sistema. En este instante haremos una copia que utilizaremos más tarde mediante la ejecución de la orden:

```
$ cp cdn.qcow2 auxsupervisor.qcow2
```

A continuación se vuelve a iniciar, pero ya sin imagen de instalación:

```
$ kvm -m 1024 -hda cdn.qcow2 -boot c -netdev user,id=user.0 -device e1000,netdev=user.0
```

Iniciado el sistema, se accede a él como administrador, utilizando las credenciales establecidas anteriormente.

Instalación de requisitos

En este momento se instalan los requisitos de los desarrollos realizados.
Los requisitos del desarrollo para el servicio de CDN son el microframework flask y la biblioteca de swift. Se ejecuta la siguiente orden para instalarlos:

```
# apt-get install python-flask python-swiftclient
```

Para cumplir con los requisitos descritos para el servidor de DNS, es necesaria la instalación de dnslib, se utilizará la utilidad pip de instalación de módulos de python:

```
# pip install dnslib
```

De mismo modo, para los requisitos del servidor DNS y el notificador es necesario instalar la biblioteca pika de python, para acceder al bus rabbitmq, ejecutando:

```
# apt-get install python-pika
```

Para configurar mediante las herramientas adecuadas que los servicios se inicien en el arranque del sistema y se paren, o notifiquen su parada, cuando el sistema se apague, es necesario instalar la herramienta chkconfig:

```
# apt-get install chkconfig
```

con ella podemos configurar que los servicios arranque en los runleveles adecuados.

También es aconsejable configurar las máquinas del entorno por su nombre en lugar de por su dirección IP, por lo que en el archivo /etc/hosts se añade la línea:

```
supervisor  150.200.190.100
```

Es importante que tengan este servidor configurado, ya que en él está keystone que es el módulo que resuelve donde están el resto de servicios.

La configuración del inicio del servidor DNS se realizará como servicio de tipo System V. Los servicios de este tipo son scripts que se ejecutan con el parámetro "start" o "stop" cuando el sistema se ejecute en el runlevel correspondiente. Dada la naturaleza del proyecto no es necesario profundizar en la naturaleza de lo runlevels, baste decir que en cada runlevel hay un conjunto de servicios que se inician o se paran, siendo estos:

runlevel 0: Apagado del sistema, se detienen todos los servicios y se apaga el sistema
runlevel 1: Monousuario sin soporte de red, modo mantenimiento donde sólo los servicios básicos están funcionando. Sólo el usuario administrador puede acceder al sistema.
runlevel 2: Igual que el runlevel 2, pero todos los usuarios pueden acceder
runlevel 3: Inicial normalmente el sistema en modo terminal.
runlevel 4: igual que el runlevel 3.
runlevel 5: igual que el runlevel 4, pero inicia también la interfaz gráfica.
runlevel 6: Reinicio del sistema, se detienen todos los servicios y se reinicia el sistema

Para iniciar el ejecutable como servicio, es necesario copiar el ejecutable al directorio /etc/init.d/. El código de los scripts de arranque se encuentra en los Anexos II, IV y V. Una vez copiados ahí, es necesario que el archivo que tener permisos de ejecución, esto se consigue mediante la ejecución de:

Para el caso del script del balanceador dns
```
# chmod +x /etc/init.d/dynamicdns
```

En el caso del notificador DNS
```
# chmod +x /etc/init.d/cdndnsnotifier.py
```

Y, por último, el caso del servicio CDN:
```
# chmod +x /etc/init.d/runcdn
```

Una vez copiados correctamente los ejecutables, se configuran para que se inicien en todos los runlevels que tengan soporte de red y se paren en todos los demás:

```
# chkconfig cdndnsnotifier.py on --level 345
# chkconfig dynamicdns.py on --level 2345
# chkconfig runcdn on --level 2345
```

A continuación se copian los ejecutables de los servicios a /usr/bin/ para que el sistema pueda encontrarlos. Los códigos se encuentran en los Anexos I y III. Una vez copiados al sistema, se copian a esa ubicación mediante las órdenes:

```
# cp cdnservice.py /usr/bin/
# cp dns.py /usr/bin/
# cp getexternalIP /usr/bin
```

Y se le establece permisos de ejecución igual que en el caso de los scripts de arranque:

```
# chmod +x /usr/bin/cdnservice.py
# chmod +x /usr/bin/dns.py.py
# chmod +x /usr/bin/getexternalIP
```

También es necesario modificar la configuración de la red para que la configuración sea automática. Esto se consigue modificando el archivo /etc/network/interfaces para que contenga las líneas:

```
auto eth0
iface eth0 inet dhcp
```

No debe contener más configuración para la interfaz eth0.

Es necesario eliminar el servicio dnsmasq para que el servidor dns puede utilizar el puerto adecuado. Se desinstala este servicio ejecutando:

```
#apt-get remove dnsmasq
```

Por último, se instala el paquete cloud-init, que permite a openstack realizar modificaciones en la instancia en el arranque de esta:

```
# apt-get install cloud-init
```

Una vez realizado esto, copiamos la imagen creada al directorio /repositorio en el servidor supervisor, siguiendo las convenciones ya establecidas en el sistema base. Este directorio donde tiene configurado la plataforma que se almacenan las imágenes de las máquinas virtuales.

Dado que la plataforma base no estaba pensada para implementar más servicios, no se dimensionó el servidor supervisor para instalar más software. Por tanto, es necesario instalar un supervisor auxiliar donde instalar los módulos de openstack de los que carece la plataforma.

Para ello, se copia el archivo antes creado auxsupervisor.qcow2 al directorio /repositorio del servidor físico.

La virtualización en este entorno se realiza mediante libvirt, por lo que es necesario definir las máquinas virtuales en este entorno. Para ello, se define el supervisor auxiliar mediante el siguiente xml, copia a /tmp/auxsupervisor.xml:

```
<domain type="kvm">
<name>auxsupervisor</name>
<memory unit="M">2048</memory>
<vcpu>2</vcpu>
<os>
<type arch="x86_64">hvm</type>
<boot dev="hd"/>
</os>
<features>
<acpi/>
<apic/>
</features>
<devices>
<emulator>/usr/bin/qemu-system-x86_64</emulator>
<disk type="file" device="disk">
<driver name="qemu" type="qcow2"/>
<source file="/repositorio/auxsupervisor.qcow2" />
<target dev="vda" bus="virtio" />
</disk>
<interface type="network">
<source network="external" />
<model type="virtio" />
<mac address="00:00:AC:15:2A:64" />
</interface>
<interface type="network">
<source network="management" />
<model type="virtio" />
<mac address="00:00:0A:05:00:64" />
</interface>
<graphics type="vnc" port="-1" />
</devices>
</domain>
```

Este máquina virtual se define con dos interfaces de red que se conectarán a las redes de administración y externa. Asignar direcciones MAC a las interfaces permite asignar direcciones IP mediante DHCP más tarde. Es importante asegurarse de que esas direcciones MAC son diferentes a las de otras máquinas en la misma red. Para ello, se ejecuta:

```
# virsh net-dumpxml external
```

cuyo resultado es:

```
<network connections='1'>
  <name>external</name>
  <uuid>f51b57cd-9c8d-4205-b4ae-0ce0c8782455</uuid>
  <forward mode='nat'>
    <nat>
      <port start='1024' end='65535'/>
    </nat>
  </forward>
  <bridge name='virbr10' stp='on' delay='0'/>
  <mac address='00:00:ac:10:2a:fe'/>
  <ip address='150.200.190.254' netmask='255.255.255.0'>
    <dhcp>
      <range start='150.200.190.250' end='150.200.190.253'/>
                <host   mac='00:00:AC:10:2A:64'   name='supervisor.external'
ip='150.200.190.100'/>
                     <host    mac='00:00:AC:10:2A:C7'    name='red.external'
ip='150.200.190.199'/>
    </dhcp>
  </ip>
</network>
```

Por lo que se verifica que la dirección MAC elegida no está ya asignada en ese segmento.

De la misma manera, se verifica para para la red management:

```
# virsh net-dumpxml management
<network connections='1'>
  <name>management</name>
  <uuid>145099a4-70c0-45b8-967c-1197d70db32d</uuid>
  <bridge name='virbr11' stp='on' delay='0'/>
  <mac address='00:00:0a:10:00:fe'/>
  <ip address='150.200.180.254' netmask='255.255.255.0'>
    <dhcp>
      <range start='150.200.180.250' end='150.200.180.253'/>
              <host   mac='00:00:0A:10:00:64'   name='supervisor.management'
ip='150.200.180.100'/>
                   <host    mac='00:00:0A:10:00:C7'    name='red.management'
ip='150.200.180.199'/>
```

```
          <host  mac='00:00:0A:10:00:65'  name='virtualizador01.management'
ip='150.200.180.101'/>
          <host  mac='00:00:0A:10:00:66'  name='virtualizador02.management'
ip='150.200.180.102'/>
    </dhcp>
  </ip>
</network>
```

Una vez asegurada la correcta configuración, se define la máquina virtual mediante la orden:

```
# virsh define /tmp/auxsupervisor.xml
```

Una vez establecidas las máquinas, es necesario modificar las redes para que les ofrezcan el servicio esperado. Es necesario modificar tres redes: data, management y network.

Para ello, ejecutamos la siguiente línea que nos permite editar la configuración de la red:

```
# virsh net-edit external
```

Y modificamos el texto para que contenga lo siguiente:

```
<network>
  <name>external</name>
  <uuid>f51b57cd-9c8d-4205-b4ae-0ce0c8782455</uuid>
  <forward mode='nat'>
    <nat>
      <port start='1024' end='65535'/>
    </nat>
  </forward>
  <bridge name='virbr10' stp='on' delay='0'/>
  <mac address='00:00:ac:10:2a:fe'/>
  <ip address='150.200.190.254' netmask='255.255.255.0'>
    <dhcp>
      <range start='150.200.190.250' end='150.200.190.253'/>
                <host   mac='00:00:AC:10:2A:64'   name='supervisor.external'
ip='150.200.190.100'/>
              <host  mac='00:00:AC:15:2A:64'  name='auxsupervisor.external'
ip='150.200.190.101'/>
                      <host    mac='00:00:AC:10:2A:C7'    name='red.external'
ip='150.200.190.199'/>
    </dhcp>
  </ip>
</network>
```

Del mismo modo, modificamos la red management, ejecutando:

```
# virsh net-edit management
```

Y estableciendo el siguiente texto:

```
<network>
  <name>management</name>
  <uuid>145099a4-70c0-45b8-967c-1197d70db32d</uuid>
  <bridge name='virbr11' stp='on' delay='0'/>
  <mac address='00:00:0a:10:00:fe'/>
```

```
  <ip address='150.200.180.254' netmask='255.255.255.0'>
    <dhcp>
      <range start='150.200.180.250' end='150.200.180.253'/>
               <host   mac='00:00:0A:10:00:64'   name='supervisor.management'
ip='150.200.180.100'/>
             <host  mac='00:00:0A:05:00:64'  name='auxsupervisor.management'
ip='150.200.180.99'/>
                    <host    mac='00:00:0A:10:00:C7'    name='red.management'
ip='150.200.180.199'/>
           <host  mac='00:00:0A:10:00:65'  name='virtualizador01.management'
ip='150.200.180.101'/>
           <host  mac='00:00:0A:10:00:66'  name='virtualizador02.management'
ip='150.200.180.102'/>
    </dhcp>
  </ip>
</network>
```

Para que estos cambios tengan efecto, es necesario reiniciar las redes. Esto se realiza ejecutando las siguientes órdenes:

```
# virsh net-destroy external; virsh net-start jmeter
# virsh net-destroy management; virsh net-start management
```

Al destruir las redes y volverlas a crear todos los nodos conectados a ellas se quedan sin conexión, por lo que es necesario reiniciar los sistemas. Esto se realiza mediante la ejecución de la orden:

```
# virsh shutdown nombremáquina; virsh start nombremáquina
```

Donde nombre máquina es el nombre de la instancia que se puede obtener de la ejecución de la orden:

```
#virsh list
```

Por tanto, es necesario ejecutar:

```
#virsh shutdown red; virsh start red
#virsh shutdown supervisor; virsh start supervisor
#virsh shutdown auxsupervisor; virsh start auxsupervisor
#virsh shutdown virtualizador01; virsh start virtualizador01
#virsh shutdown virtualizador02; virsh start virtualizador02
```

Una vez modificada la red, es necesario reconfigurar la MTU de todas interfaces que estén en la red 10.26.0.0. Esto se debe a que por esta red viajarán paquetes medinate túneles GRE. Este tipo de túneles añaden una cabecera adicional a los paquetes, haciéndo que los paquetes de 1500 bytes más la cabecera que no puedan fragmentarse no se transmitan. Tal es el caso de los paquetes de swift. Por tanto, será necesario establecer una MTU mayor ejecutando:

```
#ifconfig interfaz mtu 1800
```

En todas las interfaces desde la interfaz eth2 del servidor red a la eth0 de supervisor01, incluyendo todos los bridges e interfaces virtuales intermedias.

Instalación de módulos en supervisor auxiliar

Instalación de Swift

Swift es el módulo de almacenamiento de objetos de openstack, y el que se utilizará para almacenar los recursos de la CDN. Debido a como está repartido el espacio el espacio en el servidor, la instalación de swift se realizará en dos partes. En el servidor supervisor auxiliar se instalará el servidor de swift, que básicamente es un proxy a los contenedores reales de objetos. La segunda parte consiste en instalar los servidores de contenedores reales, que se realizará en el servidor físico, ya que es donde hay espacio libre.

Aunque no se le ha dado demasiado peso anteriormente, keystone es el módulo que registra todo los módulos, usuarios y permisos, por lo que se hará un uso intensivo de él al instalar otros módulos.

Inicialmente se cargan las credenciales de administrador de la plataforma como variables de entorno:

```
$export OS_SERVICE_TOKEN=ADMIN123456789TOKEN123456789
$export OS_SERVICE_ENDPOINT=http://172.16.0.100:35357/v2.0/
$export OS_USERNAME=admin
$export OS_PASSWORD=s3cr3t0
$export OS_TENANT_NAME=caostack
$export OS_AUTH_URL=http://150.200.190.100:35357/v2.0
```

A continuación se crea el usuario del servicio y se registra en el grupo correspondiente con los privilegios necesarios:

```
$keystone user-create --name swift --pass swift
$keystone user-role-add --user swift --tenant service --role admin
```

La configuración en la plataforma termina registrando los endpoints para que el resto de módulos puedan encontrar el servicio:

```
$keystone endpoint-create \
  --service-id $(keystone service-list | awk '/ object-store / {print $2}')
\
  --publicurl 'http://150.200.190.101:8080/v1/AUTH_%(tenant_id)s' \
  --internalurl 'http://150.200.190.101:8080/v1/AUTH_%(tenant_id)s' \
  --adminurl http://150.200.190.101:8080 \
  --region Madrid
```

Como se ha denotado anteriormente, la dirección donde se registra el servicio es el supervisor auxiliar, ya que es donde está el proxy del servicio swift que redirigirá las peticiones al nodo real de almacenamiento.

Una vez registrados todo los elementos en la plataforma, se pasa a la instalación propiamente. La instalación del software necesario se realiza ejecutando la siguiente línea:

```
# apt-get install swift swift-proxy python-swiftclient python-keystoneclient python-keystonemiddleware memcached
```

Para hacer más sencilla la instalación, es posible obtener archivos de ejemplo de configuración y posteriormente modificar sólo los parámetros necesarios.
La obtención se realiza mediante la herramienta wget:

```
$wget https://raw.githubusercontent.com/openstack/swift/juno-eol/etc/proxy-server.conf-sample
```

Y se copia el archivo al directorio de configuración de swift con el nombre adecuado:

```
# cp proxy-server.conf-sample /etc/swift/proxy-server.conf
```

En este momento es necesario editar el archivo /etc/swift/proxy-server.conf para que contenga los siguientes parámetros en las secciones adecuadas:

```
[DEFAULT]
...
bind_port = 8080
user = swift
swift_dir = /etc/swift

[pipeline:main]
pipeline = authtoken cache healthcheck keystoneauth proxy-logging proxy-server

[app:proxy-server]
...
allow_account_management = true
account_autocreate = true

[filter:keystoneauth]
use = egg:swift#keystoneauth
...
operator_roles = admin,_member_
```

```
[filter:authtoken]
paste.filter_factory = keystonemiddleware.auth_token:filter_factory
...
auth_uri = http://150.200.180.100:5000/v2.0
identity_uri = http://150.200.180.100:35357
admin_tenant_name = service
admin_user = swift
admin_password = swift
delay_auth_decision = true

[filter:cache]
...
memcache_servers = 127.0.0.1:11211
```

Es de destacar que el servidor de autenticación en el servidor supervisor y que el servicio memcached está en el propio sistema.

Una vez instalado el servidor, se pasa a instalar el nodo de almacenamiento. Aunque los apropiado es que existan varios nodos replicados utilizando la herramienta rsync, en este caso sólo existirá un nodo, por lo que no se utilizará rsync.

La instalación del software necesario para los contenedores de objetos se realiza ejecutando la siguiente línea en el servidor físico:

```
# apt-get install swift swift-account swift-container swift-object xfsprogs
```

Del mismo modo que antes, se obtienen archivos de ejemplo de configuración mediante la herramienta wget:

```
# wget -O /etc/swift/account-server.conf
https://raw.githubusercontent.com/openstack/swift/juno-eol/etc/account-
server.conf-sample

# wget -O /etc/swift/container-server.conf
https://raw.githubusercontent.com/openstack/swift/juno-eol/etc/container-
server.conf-sample
```

wget -O /etc/swift/object-server.conf
https://raw.githubusercontent.com/openstack/swift/juno-eol/etc/object-
server.conf-sample

Es necesario definir un directorio donde se almacenarán los objetos, por lo que se crea el directorio /srv/node en el servidor mediante la ejecución de la siguiente línea:

```
# mkdir -p /srv/node
```

Como se ha visto, existen tres tipos de contenedores: de objetos, de contenedores y de cuentas. Las cuentas son las cuentas de usuarios que pueden hacer uso del servicio, que pueden las mismas que gestiona keystone u otras. Los contenedores son los espacios donde se almacenan los objetos. Y, por último, los objetos son los recursos almacenados por el servicio.

La configuración de cada uno se realiza modificando del siguiente modo el archivo /etc/swift/account-server.conf:

```
[DEFAULT]
...
bind_ip = 150.200.180.254
bind_port = 6002
user = swift
swift_dir = /etc/swift
devices = /srv/node

[filter:recon]
...
recon_cache_path = /var/cache/swift
```

Es de destacar que la dirección IP en la que escucha el servicio es aquella en la que se comunica con el servidor supervisor auxiliar, que es quien le redirigirá las peticiones, así como el puerto en el que escucha, que será distinto para cada servicio, y el dispositivo donde se almacenarán los datos, que es el directorio creado anteriormente.

Del mismo modo, se modifica el archivo /etc/swift/container-server.conf

```
[DEFAULT]
bind_ip = 150.200.180.254
bind_port = 6001
user = swift
swift_dir = /etc/swift
devices = /srv/node

[filter:recon]
recon_cache_path = /var/cache/swift

E, igualmente, el archivo /etc/swift/object-server.conf

[DEFAULT]
bind_ip = 150.200.180.254
bind_port = 6001
user = swift
swift_dir = /etc/swift
```

```
devices = /srv/node

[filter:recon]
recon_cache_path = /var/cache/swift
```

Una vez configurados los servicios, es necesario definir los anillos de swift. En estos anillos se definen como se realizará la replicación de los datos de swift.

Dentro del directorio /srv/node creado deberán montar los dispositivos o volúmenes de almacenamiento. Dado que el hecho de instalarlo en esa ubicación es que no hay espacio para ubicarlo en otro sitio, lo más sencillo en crear un directorio que las veces de punto de montaje:

```
#mkdir -p /srv/node/device

También es necesario asegurarse de que swift puede acceder eset directorio
y a su directorio de caché, ejecutando las órdenes:

#chown -R swift:swift /srv/node
#chown -R swift:swift /var/cache/swift
```

Para crear los anillos, en el directorio /etc/swift del servidor físico, se ejecuta la siguiente línea:

```
$ swift-ring-builder container.builder create 10 1 1
```

Esto significa que el anillo tendrá un espacio de 2^10 particiones como máximo, se realizará una réplica de cada objeto (no tiene sentido otra cosa en este caso, ya que sólo tenemos un servidor) y se demorará al menos una hora mover una partición (conjunto de datos de swift) más de una vez.

Continuación, se añade el nodo al anillo:

```
$swift-ring-builder container.builder   add r1z1-
150.200.180.254:6001/device 100
```

Destacando que el puerto es aquel donde hemos definido que estaba el servicio adecuado, container en este caso, device es el directorio que hemos definido para almacenamiento y 100 es el peso de ese nodo en la plataforma.

Por último, se ejecuta la siguiente línea para dar el peso adecuado a cada nodo:

```
$ swift-ring-builder container.builder rebalance
```

Esta ejecución creará un archivo container.ring.gz que se utilizará más tarde.

Se puede verificar su creación ejecutando la orden:

```
$ swift-ring-builder container.builder

container.builder, build version 1
1024 partitions, 3.000000 replicas, 1 regions, 1 zones, 1 devices, 100.00
balance
The minimum number of hours before a partition can be reassigned is 1
Devices:      id   region   zone         ip address   port   replication ip
replication port      name weight partitions balance meta
                 0         1       1 150.200.180.254   6001 150.200.180.254
6001    device 100.00          1024  100.00
```

Obteniendo correctamente los datos que se han configurado.

Del mismo modo se procede con el servicio de objetos. Se crea el anillo mediante la ejecución de la siguiente línea:

```
$ swift-ring-builder object.builder create 10 1 1
```

Se añade el nodo al anillo:

$ swift-ring-builder `object.builder   add r1z1-150.200.180.254:6000/device 100`

Y se rebalancea, generándose el archivo object.ring.gz:

```
$ swift-ring-builder object.builder rebalance
```

Se puede verificar ejecutando:

```
# swift-ring-builder object.builder
object.builder, build version 1
1024 partitions, 3.000000 replicas, 1 regions, 1 zones, 1 devices, 100.00
balance
The minimum number of hours before a partition can be reassigned is 1
Devices:    id  region  zone      ip address  port  replication ip
replication port      name weight partitions balance meta
             0       1     1 150.200.180.254  6001 150.200.180.254
6001    device 100.00          1024 100.00
```

Y, por último, se procede del modo con el servicio de cuentas. Se genera el anillo mediante la ejecución de la línea:

```
$ swift-ring-builder account.builder create 10 1 1
```

se añade el nodo al anillo:

```
$ swift-ring-builder account.builder   add r1z1-150.200.180.254:6002/device
100
```

Y se rebalancea para generar el archivo account.ring.gz:

```
$ swift-ring-builder account.builder rebalance
```

Y se puede verificar mediante la ejecución de la línea:

```
$ swift-ring-builder account.builder
account.builder, build version 1
1024 partitions, 1.000000 replicas, 1 regions, 1 zones, 1 devices, 0.00
balance
The minimum number of hours before a partition can be reassigned is 1
Devices:    id  region  zone      ip address  port  replication ip
replication port      name weight partitions balance meta
             0       1     1 150.200.180.254  6002 150.200.180.254
6002    device 100.00       1024    0.00
```

Para finalizar la configuración del nodo de almacenamiento, se configura el propio servicio swift, obteniendo, como previamente, un archivo de configuración de ejemplo mediante wget:

```
#wget -O /etc/swift/swift.conf
https://raw.githubusercontent.com/openstack/swift/juno-eol/etc/swift.conf-
sample
```

Y se edita dicho archivo modificando los siguientes campos:

```
[swift-hash]
swift_hash_path_suffix = swiftsecret
swift_hash_path_prefix = swiftsecret
```

Por último, se asegura que swift puede acceder a su archivo de configuración mediante la orden:

```
# chown -R swift:swift /etc/swift
```

Una vez definido el nodo de almacenamiento, se vuelve al servidor supervisor auxiliar para realizar las últimas tareas.
Se configura el archivo /etc/swift del mismo modo que en el servidor de almacenamiento, modificando los campos:

```
[swift-hash]
swift_hash_path_suffix = swiftsecret
swift_hash_path_prefix = swiftsecret
```

A continuación se copian los archivo ring.gz generados antes del nodo contenedor al servidor auxiliar mediante scp:

```
$ scp /etc/swift/*.ring.gz auxsupervisor:/etc/swift
```

Y, por último, se reinician los servicios en el servidor supervisor auxiliar:

```
# service memcached restart
# service swift-proxy restart
```

En este momento se puede realizar la prueba de almacenar objetos mediante la ejecución de:

```
$ swift -v -V 2.0 -U service:swift -K swift upload zonaX file.txt
```

donde zonaX es el nombre del contenedor y file.txt es un archivo existente.

Y obteniéndolo de nuevo mediante:

```
$swift -v -V 2.0 -U service:swift -K swift download zonaX file.txt
```

Heat es el módulo de orquestación de openstack. Este módulo recibe información de los demás módulos y ordena ejecutar acciones en base a estos datos.
La base de datos a utilizar es la de la plataforma, ubicada en el servidor supervisor. Por tanto, los siguientes pasos serán ejecutados en el servidor supervisor.

En él ejecutamos la orden:

```
# mysql
```

y desde la consola que muestra se ejecuta:

```
CREATE DATABASE heat;
GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'localhost' \
  IDENTIFIED BY 'heat';
GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'%' \
  IDENTIFIED BY 'heat';
```

con lo que se crea la base de datos de heat y se dan los privilegios necesarios al usuario de base de datos heat. Se cierra esta consola y se sale de mysql.

A continuación, se cargan la variables de entorno de credenciales de administrador de openstack:

```
$export OS_USERNAME=admin
$export OS_PASSWORD=s3cr3t0
$export OS_TENANT_NAME=caostack
$export OS_AUTH_URL=http://150.200.190.100:35357/v2.0
```

Y se ejecuta en la consola:

```
$ keystone user-create --name heat --pass heat
$ keystone user-role-add --user heat --tenant service --role admin
$ keystone user-role-add --user heat --tenant service --role
heat_stack_owner
$ keystone role-create --name heat_stack_user
```

Con ello se crea el usuario para el servicio, se le dan los privilegios necesarios y se crea un rol para que los usuarios que puedan utilizarlo.

Una vez vez definido esto, se da de alta el servicio en la plataforma:

```
$ keystone service-create --name heat --type orchestration \
  --description "Orchestration"
```

```
$ keystone service-create --name heat-cfn --type cloudformation \
  --description "Orchestration"
```

Y, finalmente, se registran los endpoints en la plataforma para que otros módulos puedan encontrar este servicio:

```
$ keystone endpoint-create \
  --service-id $(keystone service-list | awk '/ orchestration / {print
$2}') \
  --publicurl http://150.200.190.101:8004/v1/%\(tenant_id\)s \
  --internalurl http://150.200.190.101:8004/v1/%\(tenant_id\)s \
  --adminurl http://150.200.190.101:8004/v1/%\(tenant_id\)s \
  --region Madrid

$ keystone endpoint-create \
  --service-id $(keystone service-list | awk '/ cloudformation / {print
$2}') \
  --publicurl http://150.200.190.101:8000/v1 \
  --internalurl http://150.200.190.101:8000/v1 \
  --adminurl http://150.200.190.101:8000/v1 \
  --region Madrid
```

La dirección IP utilizada es la del servidor supervisor auxiliar donde estará ubicado el servicio realmente.

Definido el servicio, se pasa instalar el servicio propiamente en el servidor supervisor auxiliar.

Para ello, se ejecuta la siguiente orden:

```
$ apt-get install heat-api heat-api-cfn heat-engine python-heatclient
```

Y se pasa a configurar el software de heat.
En el archivo /etc/heat/heat.conf se modifican los siguientes campos:

```
[database]
...
connection = mysql://heat:heat@150.200.180.100/heat
#siendo 150.200.180.100 la dirección del servidor supervisor, donde está el
servidor de base de datos.

[DEFAULT]
#rabbitmq ubicado en el servidor supervisor
rpc_backend=heat.openstack.common.rpc.impl_kombu
rabbit_host = 150.200.180.100
rabbit_password = openstackpass
rabbit_userid = openstackuser
```

```
rabbit_virtual_host=/openstack
#ubicado en el servidor supervisor auxiliar
heat_metadata_server_url = http://150.200.190.101:8000
heat_waitcondition_server_url =
http://150.200.190.101:8000/v1/waitcondition

[keystone_authtoken]
…
#keystone ubicado en servidor supervisor
auth_uri = http://150.200.180.100:5000/v2.0
identity_uri = http://150.200.180.100:35357
admin_tenant_name = service
admin_user = heat
admin_password = heat
[ec2authtoken]
...
auth_uri = http://150.200.180.100:5000/v2.0
```

Una vez configurado, se ejecuta la siguiente orden para guardar los datos necesarios en la base de datos:

```
# su -s /bin/sh -c "heat-manage db_sync" heat
```

Y, por último, se reinicia el servicio:

```
# service heat-api restart
# service heat-api-cfn restart
# service heat-engine restart
```

Opcionalmente, se puede eliminar la base de datos sqlite que se crea de forma predeterminada y no se utilizar al utilizar mysql:

```
rm -f /var/lib/heat/heat.sqlite
```

Ceilometer es el módulo de monitorización de openstack que se utiliza tanto para obtener el estado de la plataforma como para poder mantener o tarificar el uso de la plataforma. Este módulo nos sirve para nutrir al módulo Heat de información para tomar acciones.

Para iniciar la instalación, como siempre, es necesario cargar las variables de entorno de administración:

```
$ export OS_USERNAME=admin
$ export OS_PASSWORD=s3cr3t0
$ export OS_TENANT_NAME=caostack
$ export OS_AUTH_URL=http://150.200.190.100:35357/v2.0
```

Como para el resto de servicios, se crea un usuario de servicio y se le añade al tenant de servicio con el rol de administrador:

```
$ keystone user-create --name ceilometer --pass ceilometer
$ keystone user-role-add --user ceilometer --tenant service --role admin
```

Y se registra el servicio mediante la ejecución de la línea:

```
$ keystone service-create --name ceilometer --type metering \
  --description "Telemetry"
```

Por último, se termina de definir el servicio mediante la ejecución de su endpoint:

```
$ keystone endpoint-create \
  --service-id $(keystone service-list | awk '/ metering / {print $2}') \
  --publicurl http://150.200.190.101:8777 \
  --internalurl http://150.200.190.101:8777 \
  --adminurl http://150.200.190.101:8777 \
  --region Madrid
```

Siendo de destacar que la dirección IP es la del servidor supervisor auxiliar accesible desde cualquier punto de la plataforma.

La comunicación de este servicio con otras partes se realiza mediante el bus rabbitmq de la plataforma, por lo que se añade un usuario para este uso, ejecutando la línea línea en el servidor supervisor, que es donde está instalado el bus y sus herramienta de administración:

```
# rabbitmqctl add_user telemetry telemetry
```

En este punto se comienza la instalación del software del módulo en el servidor supervisor auxiliar:

```
# apt-get install ceilometer-api ceilometer-collector ceilometer-agent-
central \
  ceilometer-agent-notification ceilometer-alarm-evaluator ceilometer-
alarm-notifier \
  python-ceilometerclient mongodb-server mongodb-clients python-pymongo
```

Durante la instalación se preguntan diversos parámetros que pueden ignorarse o dejarse con el valor predeterminado, ya que posteriormente se configurarán a mano.
Ceilometer utiliza una base de datos mongodb para almacenar sus datos que es necesario configurar.
Para ello, se modifica el archivo /etc/mongodb.conf con los siguientes parámetros:

```
#interfaz IP de administración
bind_ip = 150.200.180.99
# para evitar cachés de 1G
smallfiles = true
```

Siendo la dirección IP la dirección del servidor supervisor auxiliar en la red de administración.

Una vez configurado, se inicia el servicio mediante la ejecución de la orden:
service mongodb restart

Con el servicio iniciado, se crea la base de datos de la que hará uso ceilometer. Esto se realiza ejecutando la siguiente línea:

```
# mongo --host 150.200.180.99--eval '
  db = db.getSiblingDB("ceilometer");
  db.addUser({user: "ceilometer",
  pwd: "ceilometer",
  roles: [ "readWrite", "dbAdmin" ]})'
```

Lo que devuelve:

```
MongoDB shell version: 2.4.10
connecting to: 150.200.180.99:27017/test
{
      "user" : "ceilometer",
      "pwd" : "78b0a97d1fb79f10b906c90776864b67",
      "roles" : [
            "readWrite",
            "dbAdmin"
      ],
      "_id" : ObjectId("56ae9e7e1e3098030b85ef82")
}
```

Para continuar con la configuración, lo más adecuado es generar un clave aleatoria de diez bytes mediante la orden:

```
# openssl rand -hex 10
```

En este caso, ha devuelto el valor 4c28339a850f9b0d475f

A continuación modificamos el archivo /etc/ceilometer/ceilometer.conf para que presente los siguientes valores:

```
connection=mongodb://ceilometer:ceilometer@150.200.180.99:27017/ceilometer

auth_strategy=keystone

rpc_backend = rabbit
rabbit_host = 150.200.180.100
rabbit_userid=telemetry
rabbit_password = telemetry
rabbit_virtual_host = /openstack

[keystone_authtoken]
...
auth_uri = http://150.200.180.100:5000/v2.0
identity_uri = http://150.200.180.100:35357
admin_tenant_name = service
admin_user = ceilometer
admin_password = ceilometer

[service_credentials]
...
os_auth_url = http://http://150.200.180.100:5000/v2.0
os_username = ceilometer
os_tenant_name = service
os_password = ceilometer

[publisher]
...
metering_secret = 4c28339a850f9b0d475f

[DEFAULT]
...
verbose = True
```

Es de destacar que el valor de metering_secret es el valor aleatorio antes creado, el servidor de autenticación es el servidor supervisor, las credenciales de rabbitmq son las antes creadas y el rabbit_virtual_host es el canal que ya estaba creado en el entorno.

Por último, se reinician todos los servicios de ceilometer:

```
# service ceilometer-agent-central restart
# service ceilometer-agent-notification restart
# service ceilometer-api restart
# service ceilometer-collector restart
# service ceilometer-alarm-evaluator restart
# service ceilometer-alarm-notifier restart
```

En el nodo virtualizador, que será quien instancie los nodos de la CDN, se configura la monitorización del servicio compute. Para ello, se instala el software necesario:

```
# apt-get install ceilometer-agent-compute
```

Y, en el archivo /etc/ceilometer/ceilometer.conf, se configuran los siguientes parámetros, con los mismos valores que en el servidor supervisor auxiliar:

```
[publisher]
metering_secret = 4c28339a850f9b0d475f
[DEFAULT]
rpc_backend = rabbit
rabbit_host = 150.200.180.100
rabbit_userid = telemetry
rabbit_password = telemetry
[database]
connection=mongodb://ceilometer:ceilometer@150.200.180.99:27017/ceilometer
[service_credentials]
os_auth_url = http://150.200.180.100:5000/v2.0
os_username = ceilometer
os_tenant_name = service
os_password = ceilometer
os_endpoint_type = internalURL
os_region_name = regionOne
[DEFAULT]
verbose = True
```

Para que se recojan los datos de este servidor, es necesario modificar la configuración de módulo nova en el servidor supervisor, que es donde se aloja este módulo. Los cambios necesarios son modificar los siguientes parámetros a los valores mostrados en /etc/nova/nova.conf:

```
[DEFAULT]
instance_usage_audit = True
instance_usage_audit_period = hour
notify_on_state_change = vm_and_task_state
notification_driver = messagingv2
```

Adecuación del entorno

Ha sido necesario adecuar el entorno del siguiente modo:

- ❑ Modificación de virtualizador01, estando previamente configurado con virt-manager
- ❑ Modificación de supervisor, haciendo que tenga más espacio de almacenamiento para imágenes en el servicio glance y modificando el servicio de mensajería.

Modificación de virtualizador01

En el estado inicial el sistema virtualizador01 contaba con 300MB libres de espacio de disco, 1GB de RAM y 2 CPU virtuales. Sin embargo, para cada nodo de la CDN se ha elegido asignar un GB de RAM, una CPU virtual y es necesario que se pueda albergar una copia de la imagen de disco. Como prueba, la CDN iniciará varios nodos, por lo que es necesario modificar los parámetros de esta máquina virtual. Aunque es posible cambiar algunos parámetros de una máquina virtual de libvirt en caliente, algunas modificaciones no se comportan correctamente, por lo que es mejor realizar los cambios con el sistema parado, para ello se ejecuta:

```
# virsh shutdown virtualizador01
```

Modificación de memoria y CPU virtuales
La modificación de la configuración de una máquina virtual gestionada con libvirt se realiza ejecutando:

```
# virsh edit virtualizador01
```

Esto abre un editor de texto en consola que permite modificar la configuración, se modifica esta para que presente, entre otros, los siguientes parámetros:

```
 <name>virtualizador01</name>
  <memory unit='KiB'>10485760</memory>
  <currentMemory unit='KiB'>10485760</currentMemory>
  <vcpu placement='static'>7</vcpu>
```

Esto indica que se le asignan 10GB de RAM y 7 CPU virtuales.

Una vez modificada y guardados los cambios, puede iniciarse de nuevo la máquina virtual, ejecutando la siguiente orden:

```
# virsh start virtualizador01
```

Modificación de almacenamiento en virtualizador01
La máquina virtual virtualizador01 tiene varios volúmenes de almacenamiento asignados mediante el módulo, pero estos volúmenes no están preparados dentro del sistema para ser utilizados. Dado que es necesario más espacio de almacenamiento se preparan del siguiente modo.

Existe en virtualizador01 un grupo de volúmenes denominado cinder-volumes con 25GB libres para utilizar. Esto puede verse ejecutando:

```
# vgdisplay
```

En este grupo de volúmenes se crea un volumen lógico denominado novaimages mediante la orden:

```
# lvcreate -n novaimages -L 15G cinder-volumes
```

Esto crea un volumen de 15GB denominado novaimages en el grupo cinder-volumes.
Los datos de imágenes de instancias los almacena nova en el directorio /var/lib/nova. Los siguientes pasos copian los datos de ese directorio al nuevo almacenamiento y monta el nuevo volumen bajo ese directorio.

Para empezar es necesario dar formato al nuevo espacio. Esto se consigue ejecutando:

```
#mkfs -t ext4 /dev/cinder-volumes/novaimages
```

A continuación se crea un directorio auxiliar donde montar este almacenamiento para poder copiar los datos. Dado su caracter temporal, lo adecuado es crearlo en /tmp ejecutando la siguiente línea:

```
# mkdir /tmp/aux
```

Y se monta con la siguiente orden:

```
#mount /dev/cinder-volumes/novaimages  /tmp/aux
```

Antes de movar los datos, es necesario parar el servicio nova para que no acceda a los datos:

```
# service nova-compute stop
```

y entonces se mueven los datos:

```
# mv /var/lib/nova /tmp/aux
```

mv respeta los usuario propietarios y los permisos, por lo que no es necesario tenerlo en cuenta.

A continuación se añade el nuevo punto de montaje al archivo /etc/fstab para que se monte automáticamente en el arranque. Esto se realiza editando el /etc/fstab y añadiendo la línea:

```
/dev/cinder-volumes/novaimages /var/lib/nova    ext4  defaults    0     2
```

Esta línea quiere decir que se montar el volumen /dev/cinder-volumes/novaimages en /var/lib/nova que es un sistema de archivos ext4 con parámetros predeterminados. Los dos últimos número son la frecuencia de copia de seguridad y el orden en que debe verificarlo fsck.

Modificado esto, puede desmontarse de la ubicación auxiliar, montado en su ubicación definitiva y reiniciado nova:

```
# umount /tmp/aux
# mount -a
# service nova start
```

A continuación, pueden cargarse las credenciales de openstack:

```
export OS_USERNAME=admin
export OS_PASSWORD=s3cr3t0
export OS_TENANT_NAME=caostack
export OS_AUTH_URL=http://150.200.190.100:35357/v2.0
```

y al ejecutar:

```
$nova hypervisor-show virtualizador01
```

se ven todos los recursos asignados, entre ellos se pueden observar los campos:

```
| cpu_info_topology_sockets | 7
| local_gb                  | 14
| memory_mb                 | 10000
```

con los parámetros modificados.

También es necesariomodificar el servicio de mensajería, ya que los nodos de la CDN se comunican a través, pero desde el red 150.200.190.0, cuando el servicio sólo se presta en la red 150.200.180.100. Esto se obtiene modificando el archivo /etc/rabbitmq/rabbitmw.conf, modificando la línea:

```
[{rabbit, [{tcp_listeners, [{"150.200.180.100", 5672},{"127.0.0.1",
5672]}]}]
```

a la siguiente, habilitando que funcione en ambas interfaces:

```
[{rabbit, [{tcp_listeners, [{"150.200.180.100", 5672},{"127.0.0.1", 5672},
{"150.200.190.100", 5672}]}]}]
```

Glance es el módulo de servicio de imágenes de sistemas. Esto es, permite obtener bajo demanda imágenes de discos duros para ejecutar máquinas virtuales con ellos. Nova, el módulo de computación, se comunica con glance de modo que cuando se soliciten a nova un número de instancias de un tipo de máquina virtual, es este caso de nodos cdn, nova obtendrá su configuración, donde está definido el nombre de la imagen de disco en la que se basa, en este caso cdn.qcow2. Este será el nombre de imagen que nova solicitará a glance para, una vez obtenido, iniciar una máquina virtual sobre ella.

No obstante, la plataforma estaba dimensionada para utilizar sólo unas pequeñas imágenes de ejemplo, por lo que el servidor supervisor, donde tenía glance configurado su almacenamiento, no tenía espacio suficiente para almacenar imágenes más grandes. No era posible dar más espacio al servidor supervisor, ya que todo el espacio estaba asignado al servidor físico.

La forma más sencilla que se encontró para solucionar esto fue que el sistema anfitrión sirviese mediante NFS un directorio y que el servidor supervisor lo montase.

Para realizar esta acción, se crea un directorio en el servidor físico denominado /glancespace:

```
# mkdir /glancespace
```

y se instala en el servidor el servidor de nfs mediante la orden:

```
# apt-get install nfs-server
```

A continuación se configura añadiendo la siguiente línea al archivo /etc/exports:

```
/glancespace 150.200.180.100(rw,sync)
```

siendo 150.200.180.100 la dirección IP del servidor que montará el directorio, esto es, el servidor supervisor.

Una vez hecho esto, es necesario realizar las acciones oportunas en el servidor supervisor.
Esto es, es necesario parar el servicio de glance antes de modificar su almacenamiento, mediante la orden:

```
# service glance-registry stop
# service glance-api stop
```

Posteriormente, instalar el cliente de nfs mediante la orden:

```
# apt-get install nfs-client
```

El siguiente paso es mover los datos, para lo que se crean un directorio auxiliar en /tmp:

```
# mkdir /tmp/new
```

y se monta el directorio servidor por nfs en él:

```
# mount -t nfs 150.200.180.254:/glancespace /tmp/new
```

siendo 150.200.180.254 la dirección IP del servidor de nfs, esto es, del servidor físico.

Un vez disponible el almacenamiento, se mueven los datos existentes mediante la orden:

```
# mv /var/lib/glance /tmp/new
```

siendo /var/lib/glance la ruta que tenía configurada para el almacenamiento. Mv respetará permisos e identificadores de propietario y grupo del archivo.
Y, por último se configura el archivo /etc/fstab para que se el sistema de archivos por red se monte automáticamente en el arranque. Para esto se añade la línea:

```
150.200.180.254:/glancespace  /var/lib/glance   nfs   rw,sync     0     0
```

Una vez configurado esto, se desmonta el sistema de archivos de /tmp mediante la ejecución de la línea:

```
# umount /tmp/new
```

y se monta en su nueva ubicación mediante la orden:

```
# mount /var/lib/glance
```

Si no se obtiene ningún mensaje de error es que se ha montado correctamente. En este momento deberían volver a tenerse los archivos originales en su ubicación.

Realizado el cambio, se reactiva el servicio de glance en el servidor supervisor mediante la ejecución de la orden:

```
# service glance-registry start
# service glance-api start
```

Configuración de redes

La red ya contaba con un router virtual denominado Externo que se utilizará en esta configuración. Como siempre, es necesario cargar las credenciales de usuario de openstack:

```
$export OS_USERNAME=admin
$export OS_PASSWORD=s3cr3t0
$export OS_TENANT_NAME=caostack
$export OS_AUTH_URL=http://150.200.190.100:35357/v2.0
```

La primera acción es crear una red y una subred para alojar la CDN:

```
$ neutron net-create cdn-net

$ neutron subnet-create cdn-net --name cdn-subnet –enable-dhcp --
gateway 10.233.0.1 10.233.0.0/24
```

Y, a continuación, se añade esta red al router existente

```
$ neutron router-interface-add Externo cdn-subnet
```

Dado que se va a ofrecer servicio al exterior, es necesario utilizar lo que se denominan ip flotantes. Esto es, direcciones IP de la red externa que se asocian a instancias internas, realizando la red NAT entre ellas. No es necesario crear el pool en este caso, ya que ya estaba creado en la red externa. Estas direcciones se crean ejecutando la siguiente orden, dado que red external se denomina “external”:

```
$ neutron floatingip-create external
```

Se ejecuta esta línea cinco veces, obteniendose así cinco direcciones diponibles.Se pueden observar ejecutando:

```
$neutron floatingip-list
+--------------------------------------+------------------
+---------------------+---------+
| id                                   | fixed_ip_address |
floating_ip_address | port_id |
+--------------------------------------+------------------
+---------------------+---------+
| 1422d6f3-2aa5-4bb3-a59a-5fae7108af68 |                  | 150.200.190.213
|         |
| 6d1b3f35-a97a-4bf0-b1c6-1a044de543ea |                  | 150.200.190.201
|         |
| 72bcf21d-26c4-4c72-8e17-0e976a3ae329 |                  | 150.200.190.202
|         |
| 78f774fa-029d-443b-9c7f-a5bef97da87e |                  | 150.200.190.203
|         |
| 84931f98-b8b4-4075-9240-ee1b27058a57 |                  | 150.200.190.200
|         |
+--------------------------------------+------------------
+---------------------+---------
```

Estas direcciones se asociarán más tarde a las instancias mediante la orden:

```
$ neutron floatingip-associate id_ip_flotante id_puerto_instancia
```

Donde la id del puerto de la instancia se obtiene ejecutando:

```
$neutron port-list
```

No obstante, esto no es suficiente para que el tráfico llegue a las instancias, las políticas de seguridad deben permitirlo, por lo que son necesarios los siguientes pasos:

Se crea una política para la cdn mediante la orden:

```
$ nova secgroup-create cdn "allow web and dns"
```

Y se le añaden las reglas necesarias para el tráfico http y dns:

```
$ nova secgroup-add-rule cdn tcp 80 80 0.0.0.0/0
$ nova secgroup-add-rule cdn tcp 53 53 0.0.0.0/0
$ nova secgroup-add-rule cdn udp 53 53 0.0.0.0/0
```

Será necesario establecer que las instancias pertenecen a esta política en el arranque.

Configuración de servicio

Configurados los servicios y con los servidores con las prestaciones adecuadas, se procede a configurar la plataforma.
La imagen creada para realizar las labores de nodo de servicio debe ser registrada en el módulo glance para que esté disponible para su uso. Procedemos del siguiente modo:

Se cargan las variables de entorno que son credenciales de openstack:

```
$export OS_USERNAME=admin
$export OS_PASSWORD=s3cr3t0
$export OS_TENANT_NAME=caostack
$export OS_AUTH_URL=http://150.200.190.100:35357/v2.0
```

Y a continuación se ejecuta:

```
$ glance image-create --name='cdn node' --is-public=true   --container-
format=bare --disk-format=qcow2 < /repositorio/cdn.qcow2
```

Siendo /repositorio/cdn.qcow2 la imagen copiada inicialmente.

Con la imagen del sistema ya cargada, es necesario configurar los parámetros con los que se ejecutarán las máquinas virtuales de los nodos. En openstack a esta

configuración se le denomina flavor. Existen varias configuraciones predeterminadas que son las siguientes:

```
$ nova flavor-list
+--------------------------------------+-----------+-----------+------
+-----------+------+-------+-------------+-----------+
| ID                                   | Name       | Memory_MB | Disk |
Ephemeral | Swap | VCPUs | RXTX_Factor | Is_Public |
+--------------------------------------+-----------+-----------+------
+-----------+------+-------+-------------+-----------+
| 1                                    | m1.tiny    | 512       | 1    | 0
|      | 1     | 1.0         | True      |
| 2                                    | m1.small   | 2048      | 20   | 0
|      | 1     | 1.0         | True      |
| 3                                    | m1.medium  | 4096      | 40   | 0
|      | 2     | 1.0         | True      |
| 4                                    | m1.large   | 8192      | 80   | 0
|      | 4     | 1.0         | True      |
| 5                                    | m1.xlarge  | 16384     | 160  | 0
|      | 8     | 1.0         | True      |
| 83ed1352-42ce-4d60-b814-df1a4908fe4e | proyecto.S | 128       | 2    | 0
|      | 1     | 1.0         | True      |
+--------------------------------------+-----------+-----------+------
+-----------+------+-------+-------------+-----------+
```

Por los identificadores se puede comprobar que los flavor con identificador 1 a 5 son los predeterminados del sistema, mientras que el último está creado por un usuario. Dado que el flavor m1.tiny es demasiado pequeño para los requisitos de la CDN pero el m1.small es demasiado grande, se crea un flavor a medida mediante la orden:

```
$ nova flavor-create cdnnode auto 1024 6 1
```

Esto define un flavor de 1GB de RAM, 6GB de disco libre necesario en el virtualizador y 1 procesador. En realidad no son necesarios 6GB, ya que el archivo ocupa poco más de 2GB, pero dado que el archivo qcow puede contener hasta 6GB openstack no permite levantar la máquina en otro caso.

Para poder acceder por ssh a estas instancias se configura una clave pública que se instalará en el nodo para poder acceder. En este caso se registra la del usuario user:

```
# nova keypair-add --pub-key ~/.ssh/id_rsa.pub user
```

Para finalizar, debemos saber en que redes deben acceder los nodos. En este caso deben acceder a la red externa para prestar servicio y a la red management para

acceder al servidor RabbitMQ. Para poder configurarlo es necesario conocer sus identificadores, que se obtienen ejecutando la orden:

```
$ nova net-list
+--------------------------------------+----------+------+
| ID                                   | Label    | CIDR |
+--------------------------------------+----------+------+
| 56fd0357-0e28-4591-9c91-5b0d00619c0e | cdn-net  | None |
| 6c555318-ce7f-45e3-8fe2-da6a7f10a26c | external | None |
+--------------------------------------+----------+------+
```

Configurado hasta este punto, se está en condiciones de poder lanza el primer nodo manualmente. Esto se realizan ejecutando la siguiente línea:

```
$nova boot --flavor cdnnode --image "cdn node" --nic net-id=56fd0357-0e28-
4591-9c91-5b0d00619c0e --security-group cdn --key-name user cdnnode1
```

Donde el flavor y la imagen son los configurados anteriormente, los identificadores son los de las redes external y producción, la key es la clave ssh antes registrada y el parámetro final es el nombre dado a la instancia. El nombre puede repetirse, ya que la instancia está identificada por un identificador único automático.

Si la ejecución funciona, es posible ver el resultado mediante la orden:

```
$nova list
| ID                                   | Name           | Status  | Task
State | Power State | Networks
| 5d86de41-8d78-4e30-bfb4-479df8bd33ad | cdnnode1       | ACTIVE  | -
| Running     | produccion=10.223.0.2 |
+--------------------------------------+----------------+---------
+------------+-------------
+-----------------------------------------------------+
```

Aumento de instancias bajo demanda

Para el aumento de instancias de nodos de la CDN en función de la demanda, es necesario configurar heat mediante un plantilla que inicie instancias mediante nova en función de los datos recopilados por ceilometer.

Inicialmente es necesario definir una entidad sobre la que trabajar. En este caso se denomina CDN, y que almacena en CDN.yml. Este archivo representa a los nodos a monitorizar, generándose el archivo CDN.yml con el siguiente contenido:

```
heat_template_version: 2013-05-23
description: CDN node
resources:
  server:
    type: OS::Nova::Server
    properties:
      image: cdn node
      flavor: cdnnode
      security_groups:
        - cdn
      networks:
        - network : cdn-net

  floating_ip:
     type: OS::Neutron::FloatingIP
     properties:
       floating_network_id: 6c555318-ce7f-45e3-8fe2-da6a7f10a26c

  association:
     type: OS::Neutron::FloatingIPAssociation
     properties:
       floatingip_id: { get_resource: floating_ip }
       port_id: {get_attr: [server, addresses, cdn-net, 0, port]}
```

Donde la red cdn-net es la red definida para la CDN y el valor de floating_network_id es el identificador de la red externa, obtenido de ejecutar:

```
$ neutron net-list
+--------------------------------------+----------
+------------------------------------------------------+
| id                                   | name     | subnets
|
+--------------------------------------+----------
+------------------------------------------------------+
| 56fd0357-0e28-4591-9c91-5b0d00619c0e | cdn-net  | 9fa02212-de56-4623-
8346-4a02ac417050 10.233.0.0/24    |
| 6c555318-ce7f-45e3-8fe2-da6a7f10a26c | external | d9b85eff-b01b-4279-
b517-07d9149a4cac 150.200.190.0/24 |
+--------------------------------------+----------
+------------------------------------------------------+
```

Una vez definido este recurso, se define la plantilla que establece la acción de escalar automáticamente. La siguiente plantilla establece que se debe añadir un nodo más, con un máximo de cinco, en caso de que el uso de cpu supere el umbral de 50% en un periodo de 15 segundos, hasta un máximo de 5 instancias. Estos valores son adecuados para una prueba de 10 minutos. No obstante, para un entorno de producción sería recomendable aumentarlos, ya que la propia monitorización también carga el sistema.
Se crea el archivo autoscale.yaml con el siguiente contenido:

```
heat_template_version: 2013-05-23
description: A simple auto scaling group.
resources:
  group:
    type: OS::Heat::AutoScalingGroup
    properties:
      cooldown: 60
      desired_capacity: 2
      max_size: 5
      min_size: 1
      resource:
        type: OS::Nova::Server::CDN

  scaleup_policy:
    type: OS::Heat::ScalingPolicy
    properties:
      adjustment_type: change_in_capacity
      auto_scaling_group_id: { get_resource: group }
      cooldown: 60
      scaling_adjustment: 1

  cpu_alarm_high:
    type: OS::Ceilometer::Alarm
    properties:
      meter_name: cpu_util
      statistic: avg
      period: 15
      evaluation_periods: 1
      threshold: 50
      alarm_actions:
        - {get_attr: [scaleup_policy, alarm_url]}
      comparison_operator: gt
```

Basada en la plantilla de http://superuser.openstack.org/articles/simple-auto-scaling-environment-with-heat

También es necesario que exista el archivo environment.yaml con el siguiente contenido para que se pueda encontrar la plantilla anterior en el espacio de nombres:

```
resource_registry:
    "OS::Nova::Server::CDN": "CDN.yaml"
```

Por último, se lanza la orquetación con la orden:

```
$ heat stack-create autoscale -f autoscale.yaml -e environment.yaml
```

Esto creará las dos primeras instancias del grupo, tal y como define la plantilla. como número deseado, como puede ver mediante la ejecución de la orden:

```
$ nova list
+------------+-------------+--------------------------------------+
```

```
| ID                                   | Name
| Status | Task State | Power State | Networks
|+------------+-------------+-----------------------------------+
| 86d319d1-396e-450a-aa99-fccee76ec8ac | au-qr7s-bxfwy4hqpzpv-vbmpc54rdocw-
server-ekwfx536zryl | ACTIVE | -          | Running     | cdn-
net=10.233.0.25, 150.200.190.205 |
| 4804cba5-ed07-46d9-9049-890e97648dee | au-qr7s-wfzqjcc2yhse-flybjnr63jdf-
server-u7krno3fom7n | ACTIVE | -          | Running     | cdn-
net=10.233.0.24, 150.200.190.204 |
```

Con este entorno en funcionamiento, es posible pasar a la fase de pruebas.

Pruebas

Para poder realizar pruebas de carga es necesario montar un entorno de pruebas y definir un protocolo de pruebas. Para realizar esta tarea se ha optado por la herramienta Jmeter de la fundación Apache. Sobre se define un plan de pruebas que permite realizar pruebas de carga

Para la realización de pruebas de carga se utiliza la herramienta jmeter de la fundación Apache.

Esta herramienta utiliza los siguientes conceptos:

- ❑ workbench como agrupador de los demás elementos.
- ❑ Test plan que es un conjunto de pruebas
- ❑ Threads que permiten realizar las mismas acciones en paralelo, simulando varios sistemas concurrentes.
- ❑ Pruebas que son peticiones que se realizan dentro de un plan de pruebas.

Si bien existen también sentencias de control, formas de comprobación de las respuestas de las peticiones y bucles, ya que sólo se desea generar carga, no se entrará en detalle.
El plan de pruebas a realizar es solicitar cinco objetos distintos desde varios usuarios concurrentes. Si bien debería utilizarse un tiempo de espera entre peticiones de tipo Poisson para simular las peticiones de usuario y el mismo tipo de espera entre el inicio de peticiones de un usuario y el siguiente, una espera lineal permite caracterizar de forma más determinista el comportamiento del sistema.

Instalación
Es necesario tener instalado el entorno JRE de java para que Jmeter pueda ejecutarse.
Con el entorno Java instalado, se descarga el archivo de Jmeter de la siguiente dirección, mediante la orden:

```
$ wget http://apache.rediris.es//jmeter/binaries/apache-jmeter-2.13.zip
```

Y se descomprime en el mismo directorio:

```
$ unzip apache-jmeter-2.13.zip
```

Esto creará un directorio apache-jmeter-2.13, dentro del cual existe otros directorio bin, se accede a él mediante:

```
$ cd apache-jmeter-2.13/bin
```

Dentro de este directorio existe un archivo jmeter al que es necesario dar permisos de ejecución para poder utilizar. Esto se realiza ejecutan la siguiente línea:

```
$ chmod +x jmeter
```

Esto abrirá su interfaz gráfica:

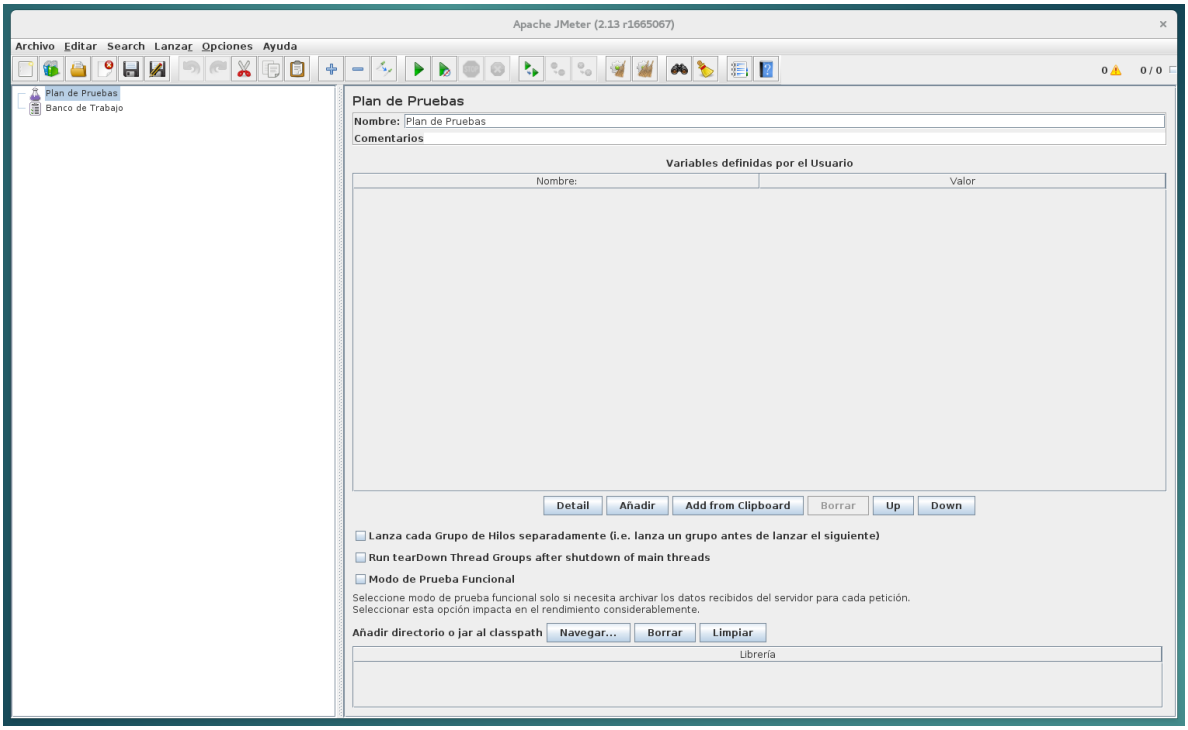

Figura 8: Interfaz gráfica Jmeter

Pulsando sobre el icono “plan de pruebas” se establecen los parámetros de la prueba. Para ello, se pulsa sobre el botón “añadir” para añadir cada campo:

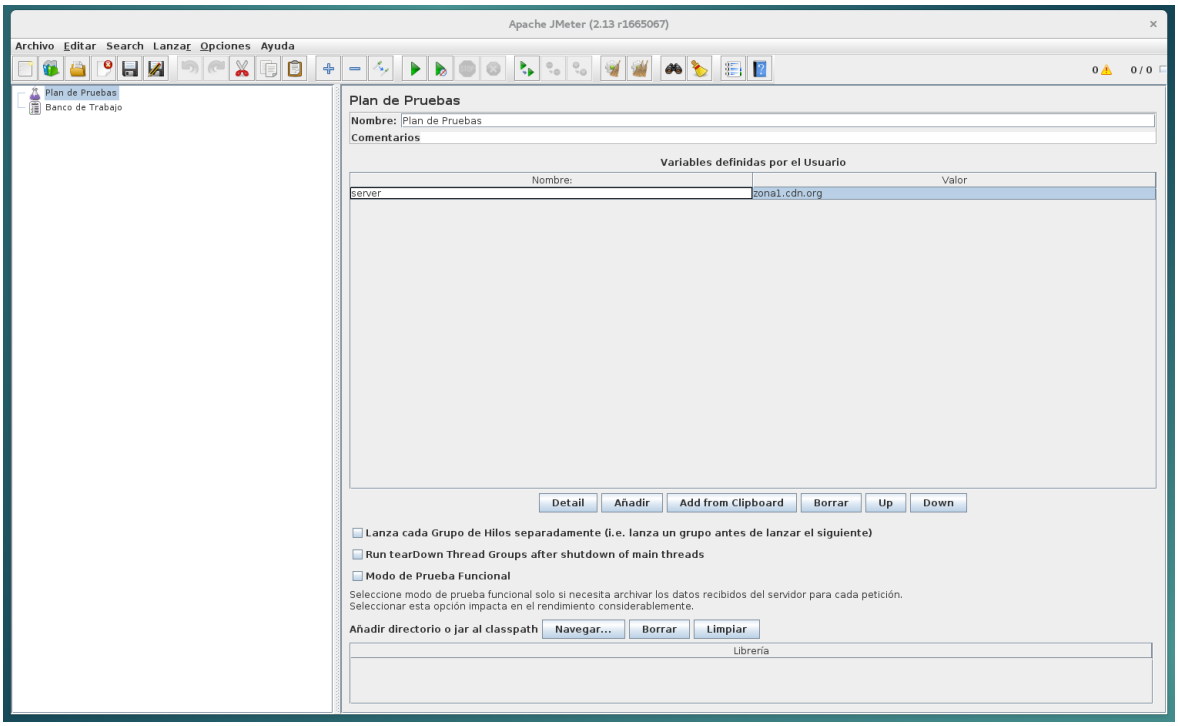

Figura 9: Configuración primer parámetro Jmeter

Del mismo modo se añaden y establecen los siguientes parámetros:

Nombre	Valor	Descripción
server	zone1.cdn.org	Dominio que resuelve el servidor DNS
client	client1	Nombre del cliente/contenedor a utilizar
object1	logo1.png	Nombre del objeto de la petición 1
object2	text1.txt	Nombre del objeto de la petición 2
object3	logo2.png	Nombre del objeto de la petición 3
object4	text2.png	Nombre del objeto de la petición 4
object5	logo3.png	Nombre del objeto de la petición 5
samplewait	100	Tiempo de espera entre peticiones en ms
upliftingtime	600	Tiempo en segundos hasta alcanzar el máximo de usuarios concurrentes
users	6	Número máximo de usuarios concurrentes

De este modo, se añadirá un usuario cada 60 segundos, existiendo un usuario el primer minuto, dos usuarios el segundo minuto y del mismo modo hasta seis usuarios. Como se hace una peticiones cada 100 ms, con un usuario serán diez peticiones por segundo, con dos usuarios veinte peticiones por segundo hasta

alcanzar las 60 peticiones por segundo. Dado que el cambio de régimen de peticiones se realiza cada 60 segundos, es tiempo suficiente para monitorizar los cambios en la plataforma. El resultado final es el siguiente:

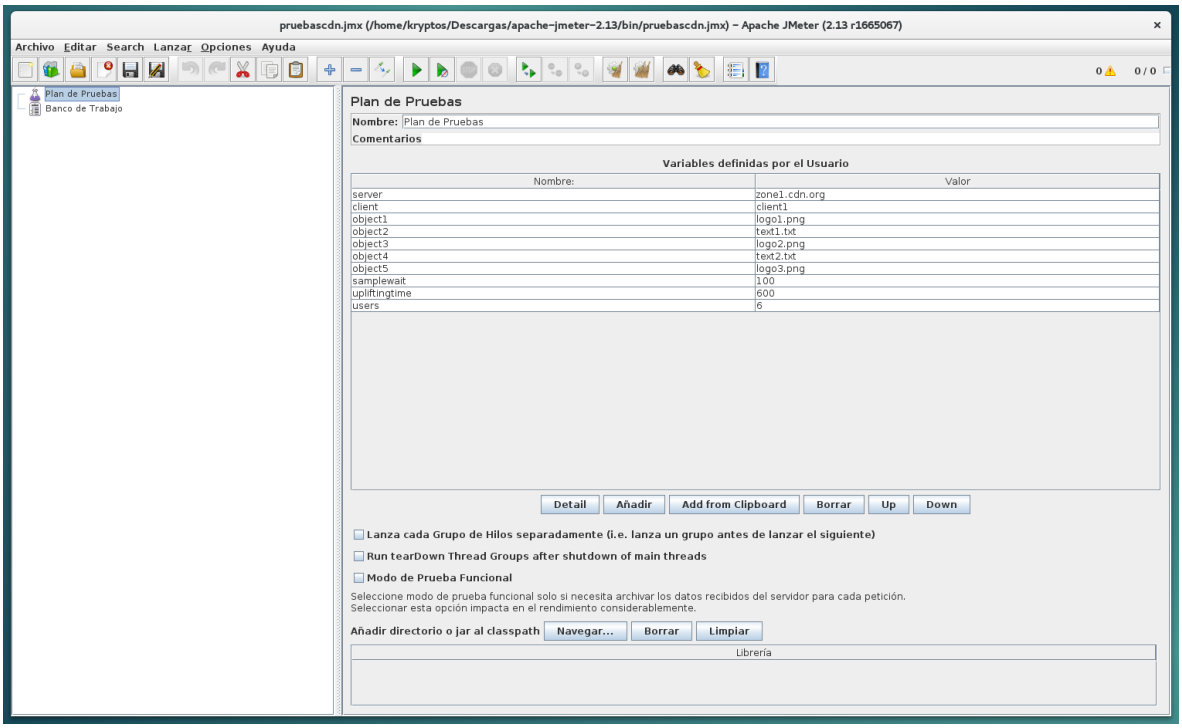

Figura 10: Configuración parámetros de prueba Jmeter

Pulsando con el botón derecho en “Plan de pruebas”, se añade un grupo de hilos siguiendo los menús que se muestran en la figura:

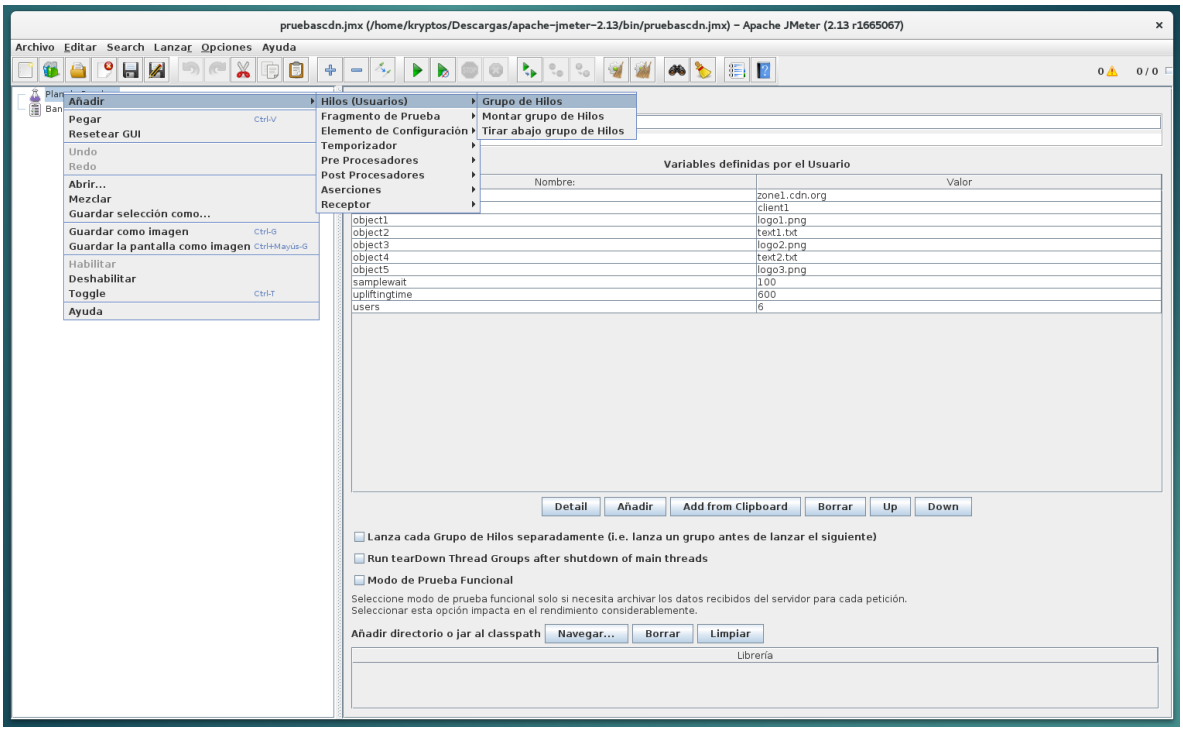

Figura 11: Adición de hilos concurrentes Jmeter

Una vez añadido, se configura para utilizar los parámetros antes establecidos:

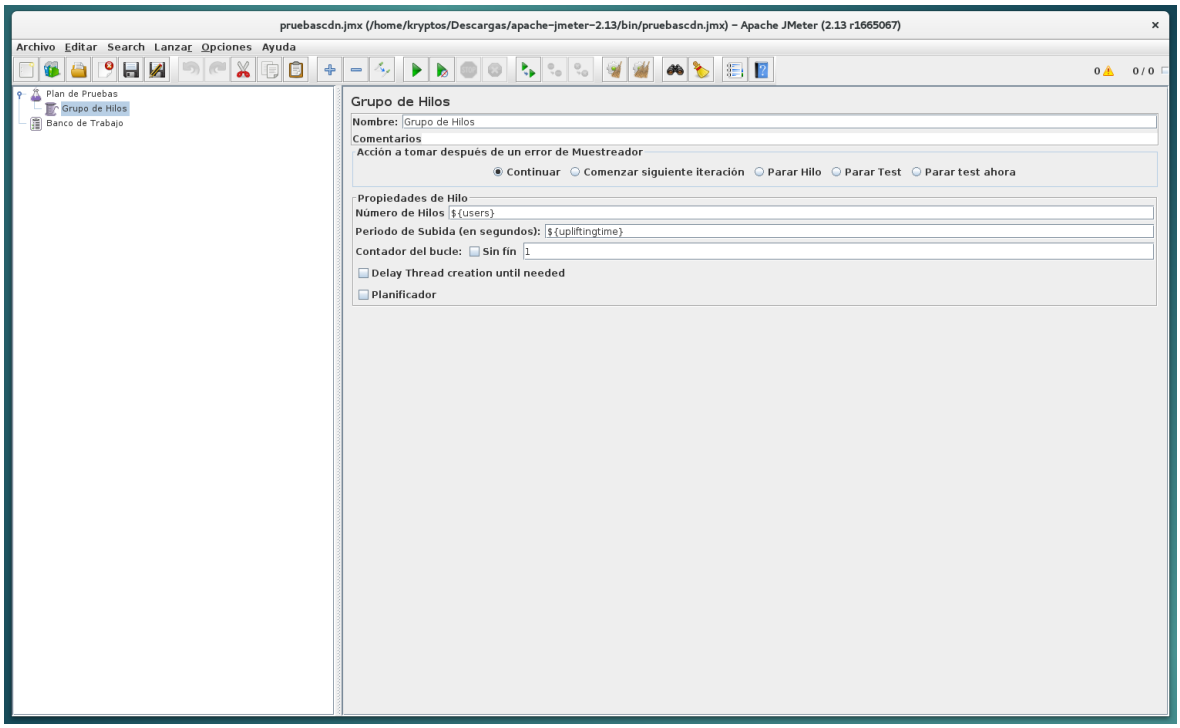

Figura 12: Configuración de hilos concurrentes Jmeter

Estableciendo el número de hilos al valor ${users} y el tiempo de subida a ${upliftingtime}.

A continuación, se añade un bucle en el que se repetirán las peticiones durante la duración de la prueba. El bucle se añade pulsando con el botón derecho en el grupo de hilos y seleccionando los menús mostrados:

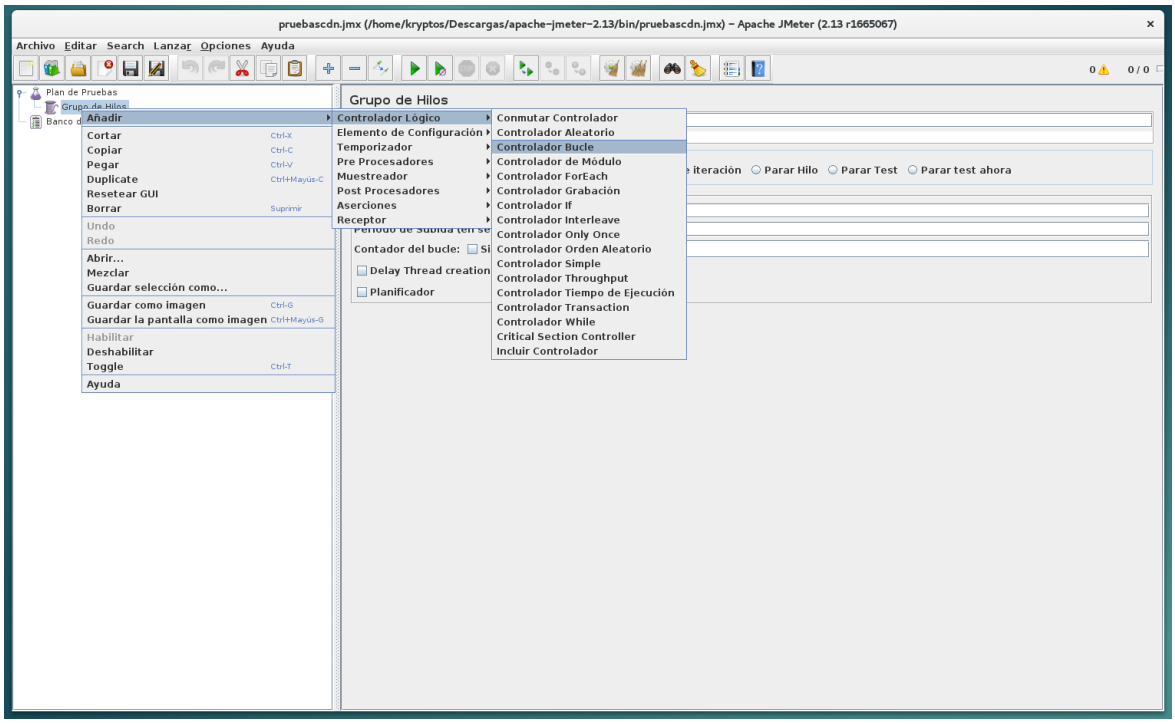

Figura 13: Adición de bucle Jmeter

En este momento se configura el número de iteraciones del bucle en el siguiente campo:

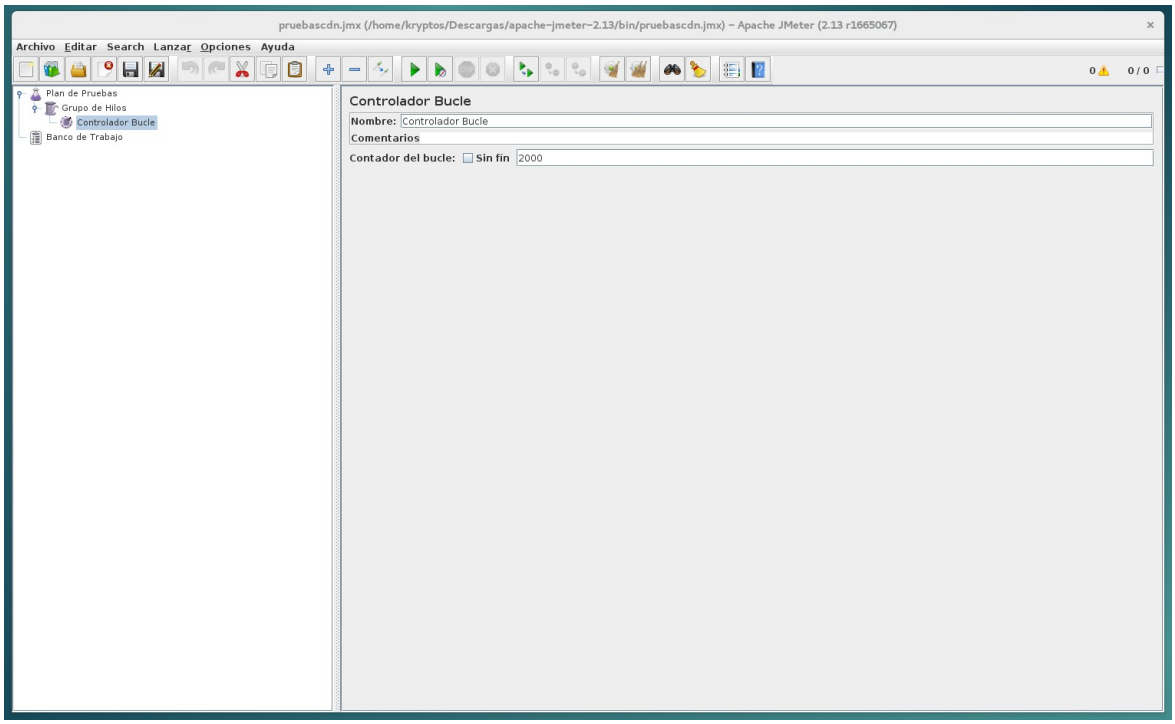

Figura 14: Configuración bucle Jmeter

Dado que se muestrea cada 100 ms, una iteración son 500 ms. Es necesario que la simulación dure más de los 10 minutos configurados para la entrada de usuarios, por lo que 2000 iteraciones es suficiente, al tomar 1000 segundos, lo que son 16 minutos y medio. Tiempo suficiente para que conocer el comportamiento de la plataforma.

Dentro del bucle se añade un muestreador HTTP, pulsando con el botón derecho en el elemento del bucle y siguiendo los menús que se muestran en la figura:

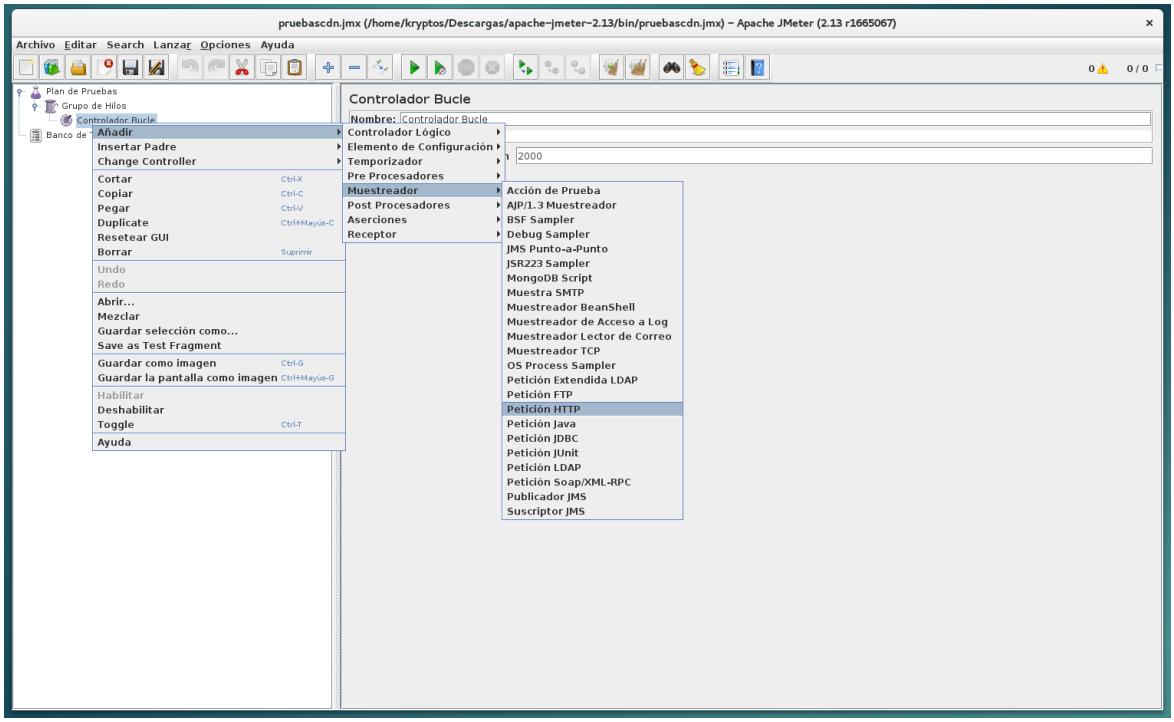

Figura 15: Adición de muestreador Jmeter

Se configura el muestreador con los parámetros configurados al inicio para el servidor, el cliente y el objeto a obtener:

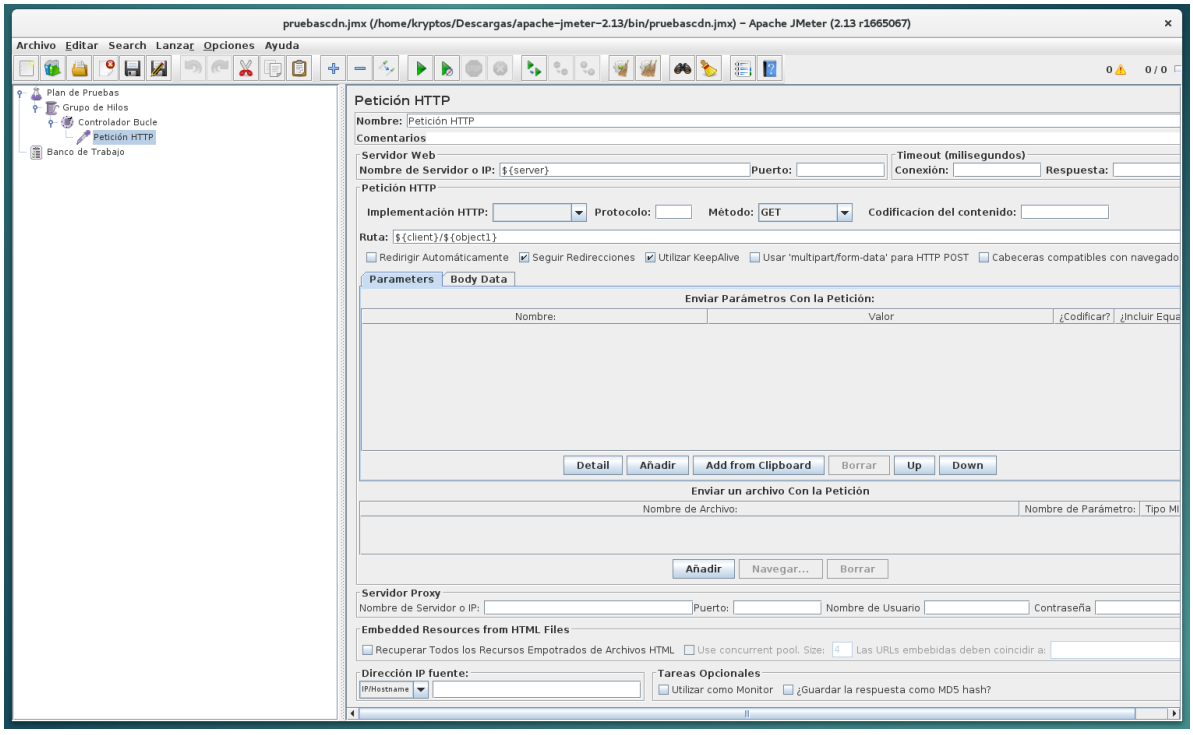

Figura 16: Configuración muestreador Jmeter

Esto es, se utiliza la variable ${server} en el nombre del servidor y se configura ${client}/$object en el campo ruta.

Siguiendo el mismo procedimiento, se añade una acción de prueba que se utiliza a modo de retardante bajo el elemento bucle con el siguiente menú:

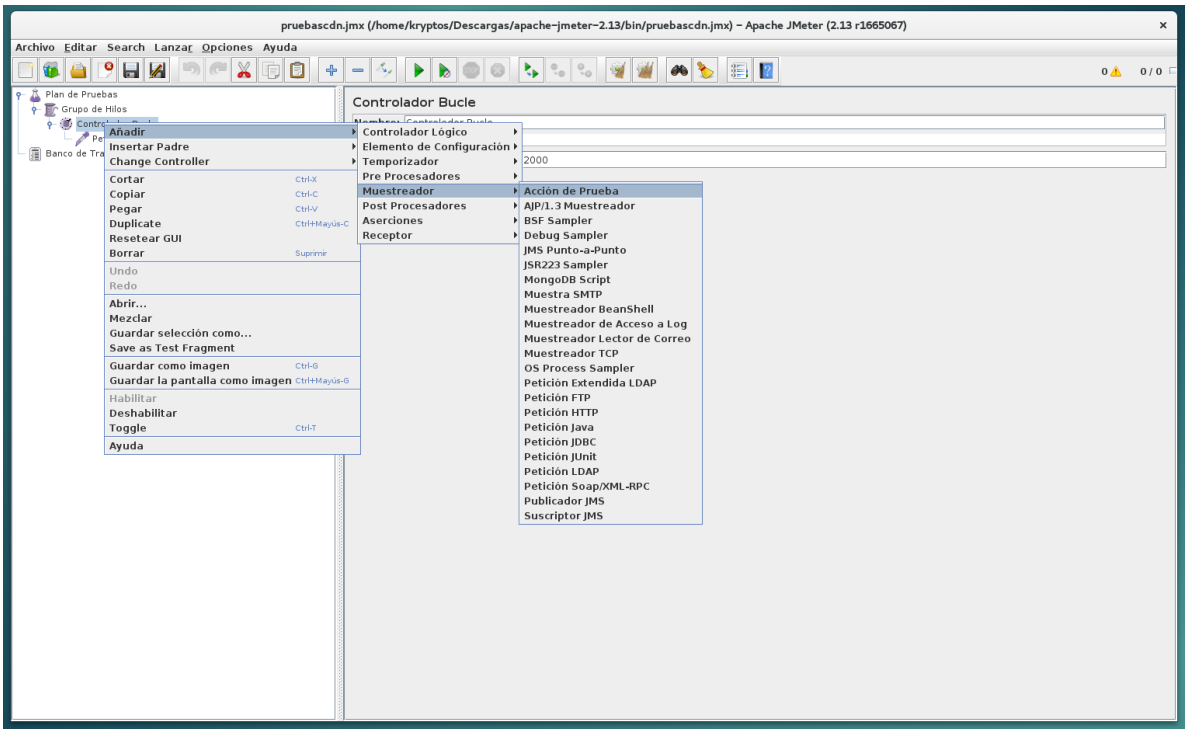

Figura 17: Adición de muetreador retardante Jmeter

Y se configura el tiempo de retardo:

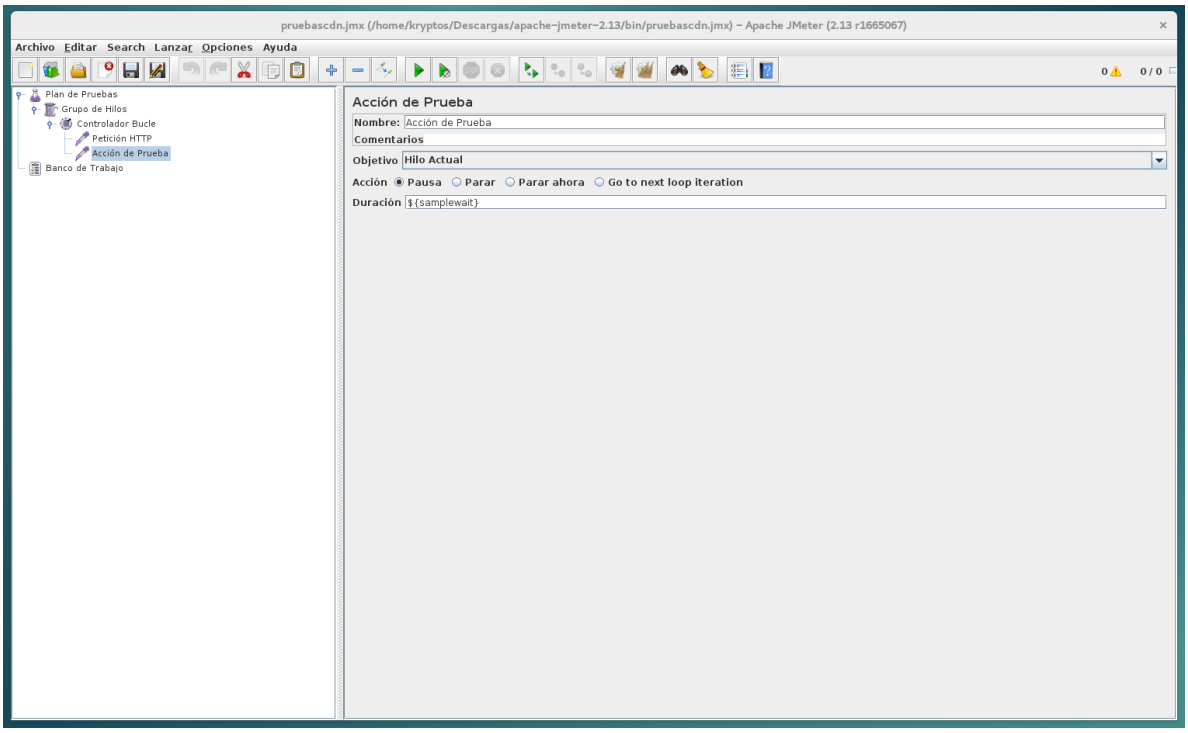

Figura 18: Configuración muestreador retardante

Esto se realiza configurando la variable ${samplewait} en el campo duración.

Se sigue el mismo procedimiento para añadir otros muestreadores y retardos hasta obtener la siguiente configuración:

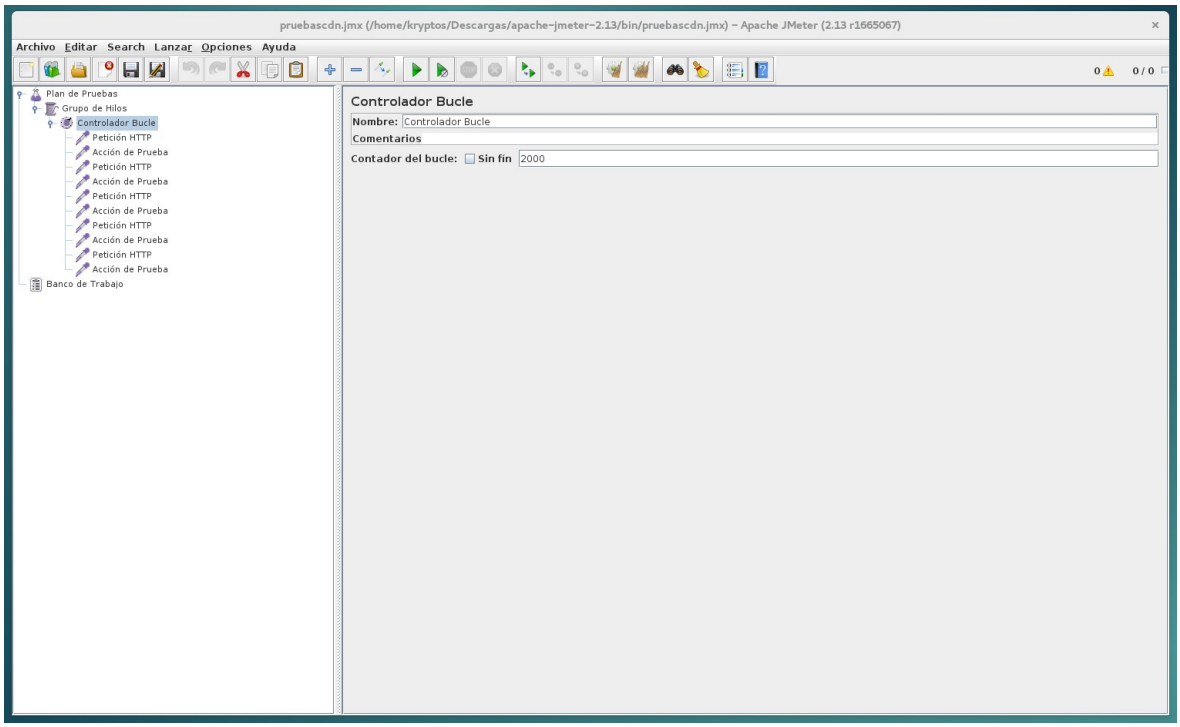

Figura 19: Bucle con todos los elementos Jmeter

Configurando en cada petición HTTP que se obtenga un objeto distinto, de object1 a object5.

Por último se guarda el plan de pruebas usando el siguiente menú:

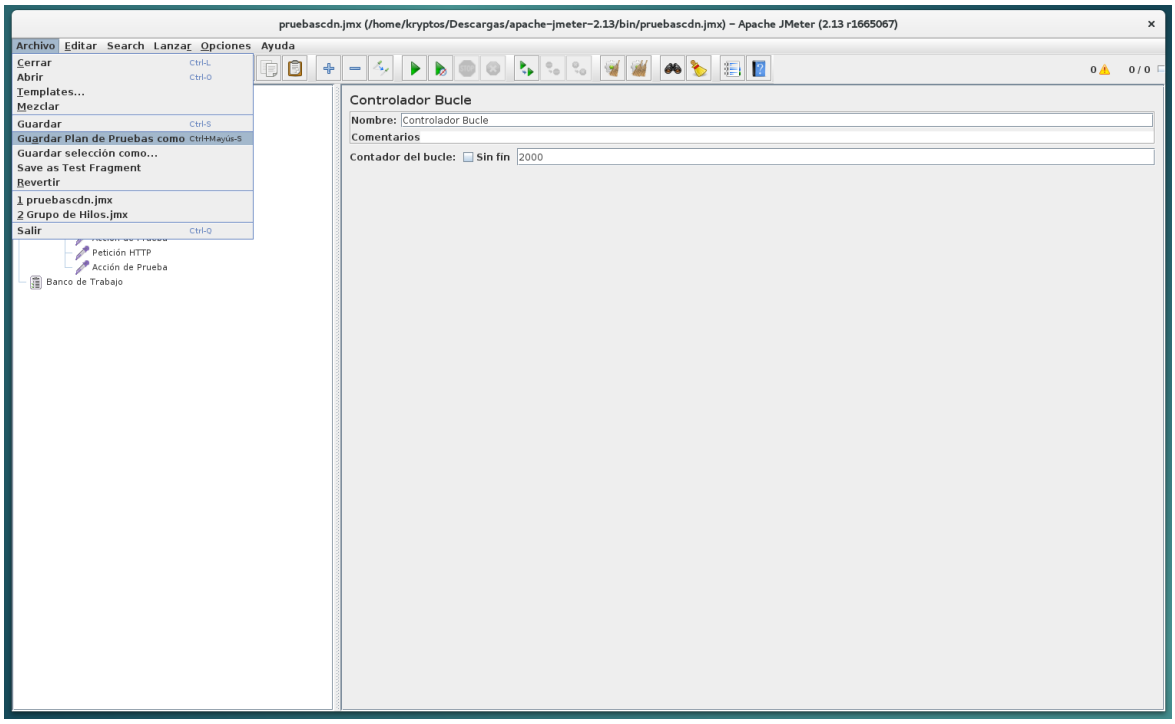

Figura 20: Guardado plan de pruebas Jmeter

Guardando el plan de pruebas con el nombre pruebascdn.jmx. Es necesario copiar este archivo al sistema desde donde se vaya a generar el tráfico para la prueba.

Capítulo 5 - Pruebas

Entorno de pruebas

El entorno se realizará con dos instancias de nodos de servicio y una instancia de nodo DNS para el envío de objetos.

Protocolo de pruebas de balanceador DNS

Entorno: Sistema con balanceador DNS ejecutándose y conectado al servidor RabbitMQ. Cliente de RabbitMQ que puede enviar mensajes.

Prueba #1 Añadir nodo nuevo a zona existente
Datos necesarios: Dirección de nodo no existente y zona existente
Realización: envío de mensaje “add direccionIP zona”
Resultado esperado: Se muestra en el registro que se ha añadido el nodo a la zona
Resultado obtenido: OK

Prueba #2 Añadir nodo existente a zona existente
Datos necesarios: Dirección de nodo existente y zona existente
Realización: envío de mensaje “add direccionIP zona”
Resultado esperado: No se modifica la zona. El nodo sólo aparece una vez.
Resultado obtenido: OK

Prueba #3 Añadir nodo a zona no existente
Datos necesarios: Dirección de nodo no existente y zona no existente
Realización: envío de mensaje “add direccionIP zona”
Resultado esperado: Se crea la zona con el nodo asociado
Resultado obtenido: OK

Prueba #4 Eliminar nodo existente en zona existente
Datos necesarios Dirección de nodo existente y zona existente
Realización: envío de mensaje “del direccionIP zona”
Resultado esperado: Se muestra en el registro que se ha eliminado el nodo de la zona
Resultado obtenido: OK

Prueba #5 Eliminar nodo no existente de zona existente
Datos necesarios: Dirección de nodo no existente y zona existente
Realización: envío de mensaje “del direccionIP zona”
Resultado esperado: No varía la zona

Resultado obtenido: OK

Prueba #6 Eliminar el último nodo de zona existente
Datos necesarios: Dirección de nodo existente y zona existente
Realización: envío de mensaje “del direccionIP zona”
Resultado esperado: Se elimina la zona
Resultado obtenido: OK

Prueba #7 Eliminar nodo de zona inexistente
Datos necesarios: Dirección de nodo no existente y zona existente
Realización: envío de mensaje “del direccionIP zona”
Resultado esperado: El registro no varía
Resultado obtenido: OK

Prueba #8 Balanceo de carga
Datos necesarios: dirección de zona en que se presta servicio con varios nodos
Realización: ejecución de dig @servidorDNS direcciondezona repetidas veces
Resultado esperado: La resolución de la zona itera sobre los nodos existentes.
Resultado obtenido: OK

Protocolo de pruebas de servicio de CDN

A continuación se detalla el protocolo de pruebas a realizar.

Prueba #1 Solicitud de objeto existente
Datos necesarios: Nombre de objeto y de cliente existentes y dirección de la CDN
Realización: Solicitud HTTP con wget a la URL definida por la
http://direccion/cliente/objeto
Resultado esperado: Respuesta HTTP con el objetos esperado
Resultado obtenido: OK

Prueba #2 Solicitud de objeto inexistente
Datos necesarios: Nombre de objeto y de cliente existentes y dirección de la CDN
Realización: Solicitud HTTP con wget a la URL definida por la
http://direccion/cliente/objeto
Resultado esperado: Error 404
Resultado Obtenido: OK

Protocolo de pruebas notificador

Entorno: Servidor con el servidor DNS funcionando y conectado a servidor rabbitMQ, y servidor con el notificador instalado.

Prueba #1 Registro en el servidor DNS
Datos necesarios: Dirección IP del nodo a registrar, zona en la que registrar. Se configura el nodo para esta zona.
Realización: ejecución de la orden “/etc/init.d/cdndnsnotifier.py start”
Resultado esperado: Se registra la dirección IP del nodo en la zona en el servidor DNS
Resultado obtenido: OK

Prueba #2 Registro en el servidor DNS
Datos necesarios: Dirección IP del nodo a dar de baja, zona en la que está registrado.Realización: ejecución de la orden “/etc/init.d/cdndnsnotifier.py stop”
Resultado esperado: Se da de baja la dirección IP del nodo en la zona en el servidor DNS
Resultado obtenido: OK

Realización de pruebas de carga

Preparación de entorno
Es necesario instalar obtener y descomprimir Jmeter en una máquina con acceso a la red externa de la infraestructura. Al mismo tiempo, es necesario abrir una consola en la máquina física. Es necesario que se hayan almacenado objetos con los nombres configurados en el plan de pruebas.

Además, para realizar estas pruebas, dado el plan de pruebas realizado, es necesario rebajar los periodos de muestreo, de forma que los cambios sean percibidos cuanto antes por la plataforma. Para esto, se modifican los siguientes archivos:

En el servidor auxsupervisor se modifca el archivo /etc/ceilometer/ceilometer.conf con el siguiente contenido:

```
[alarm]
evaluation_interval=20
```

De este modo se evalúan las alarmas cada 20 segundos, y se reinicia el servicio ejecutando:

```
#service ceilometer-agent-central restart
#service ceilometer-agent-notification restart
#service ceilometer-api restart
```

```
#service ceilometer-collector restart
#service ceilometer-alarm-evaluator restart
#service ceilometer-alarm-notifier restart
```

Por otro lado, se modifica el archivo /etc/ceilometer/pipeline.yaml en los servidor virtualizador01 y virtualizador02, para que contenga lo siguiente:

```
sources:
   - name: meter_source
     interval: 15
     meters:
         - "*"
     sinks:
         - meter_sink
   - name: cpu_source
     interval: 15
     meters:
         - "cpu"
```

De este modo se monitoriza la CPU cada 15 segundos.

Una vez realizado esto, en la máquina física se cargan las credenciales de usuario de openstack y se ejecuta:

```
$ watch -n 1 “nova list; date”
```

Lo que mostrará el número de nodos ejecutándose en cada momento y la hora en que se ha obtenido la muestra.
Desde la máquina con Jmeter, se ubica en el directorio bin de jmeter y se ejecuta:

```
$ jmeter -n -t pruebascdn.jmx
```

En ese momento jmeter comenzará a generar carga y se podrá ver la evolución de la plataforma. Conocido que se ha definido una carga de un usuario cada minuto y conocido el retardo entre peticiones, es posible saber con cuanta carga la infraestructura levanta nodos nuevos. Para que la prueba funcione correctamente es necesario modificar el servidor dns del sistema desde donde se lance la ejecución. Esto se realiza editando el archivo /etc/resolv.conf y añadiendo la siguiente línea al principio:

```
nameserver direcciónDNS
```

donde direcciónDNS es la dirección IP del nodo que funciona como balanceador DNS.

Capítulo 6 - Planificación

Desglose de tareas

Planificación			
Fase	Tarea	Carga (Jornadas/hombre)	Duración (Jornadas)
Análisis	Planteamiento	5	10
	Análisis de infraestructura existente	5	10
	Descomposición de Módulos e interfaces	3	3
	Descomposición de fases	2	2
	Análisis Legal	2	2
Servicio CDN	Diseño	2	4
	Selección de tecnologías	1	2
	Desarrollo	2	2
	Pruebas	1	1
Balanceador DNS	Diseño	2	4
	Selección de tecnologías	2	4
	Desarrollo	3	3
	Pruebas	1	1
Adecuación de Entorno	Creación de imagen de nodo CDN	3	3
	Modificación de servidores existentes	3	3
	Instalación de nuevos módulos Openstack	5	5
Configuración de servicio	Configuración de servicio y orquestación	3	3
	Pruebas finales	1	1
Documentación	Documentación del proyecto	4	63
Total (Jornadas)		46	
Total (Horas)		368	

Diagrama de Gantt

A continuación se muestra el diagrama de Gantt, de donde se deduce que el proyecto es ejecutado es 48 jornadas.

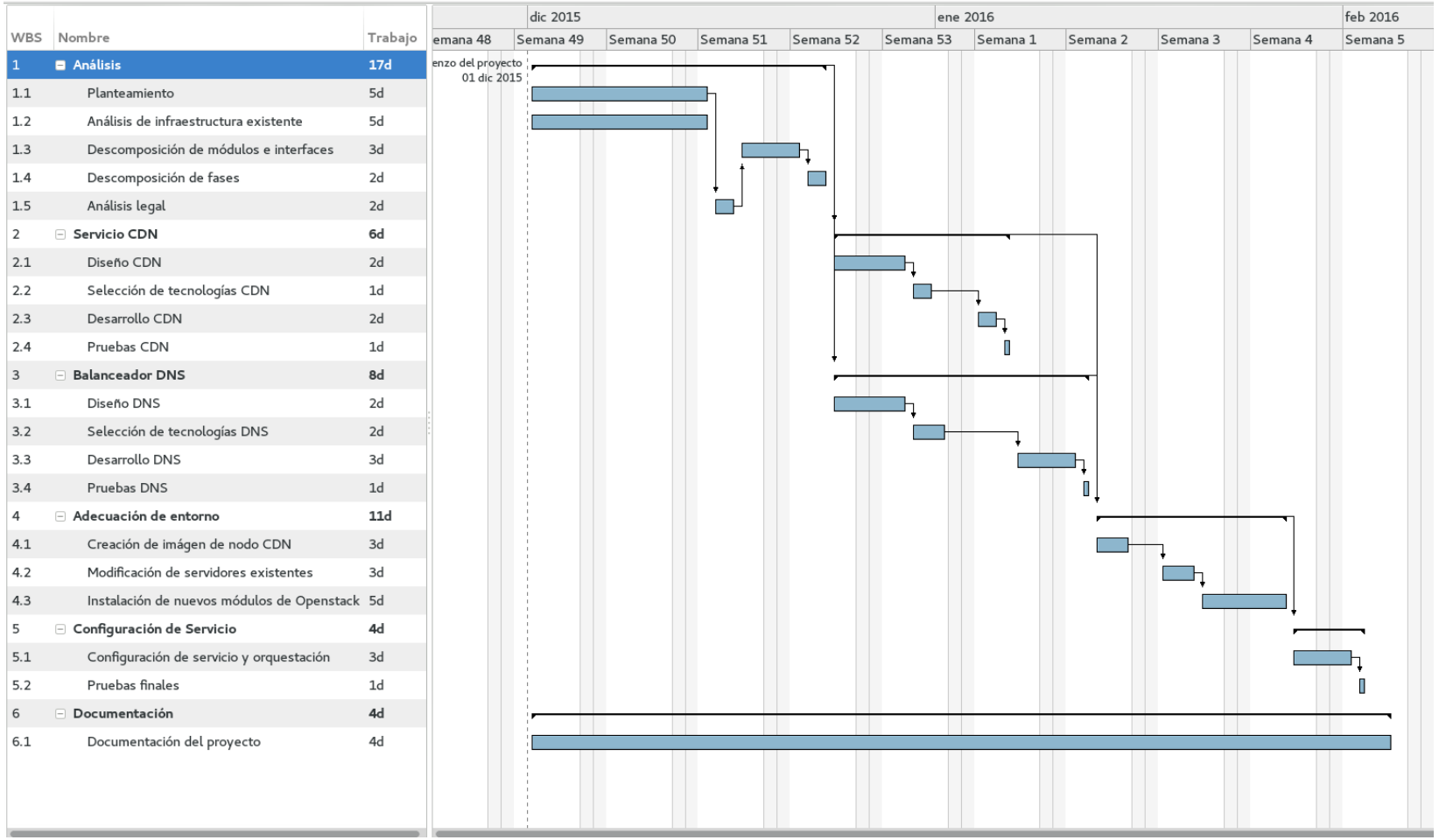

Figura 21: Diagrama de Gantt

Capítulo 7 - Marco regulador

No existe un marco regulador sobre la gestión de contenidos en España o en el marco de la unión europea. No obstante, la utilización de redes de contenido, que permiten una entrega de datos a los usuarios más rápida para aquellos generadores de contenidos que puedan pagar sus servicios, siendo las redes de contenidos a su vez clientes de los propios operadores, pueden enmarcarse dentro de la neutralidad en la red, que actualmente está siendo objeto de debate dentro de la “European 2020 Initiative”[16].

A modo de ejemplo puede analizarse el caso de Netflix[17] en España. Netflix es un proveedor de contenidos que ofrece la visualización de series y películas bajo demanda a sus suscriptores de pago, con lo que tiene red de contenidos propia. Para poder prestar el servicio de streaming adecuadamente, evitando latencias y jitter, necesita enviar los datos a los usuarios lo más próximo posible a ellos.
Su buena publicidad hace que algunos proveedores de servicios de internet admitan un acuerdo de “peering” con ellos, esto es, que el intercambio de tráfico sea gratuito. No obstante, existen proveedores de servicios de internet cuyo modelo es que cualquiera que desee ofrecer un mejor servicio a sus usuarios, para lo que deben conectarse directamente a
sus redes, debe abonar una cantidad. Este es el caso de Movistar en España.
En este escenario se dan los siguientes casos:

- ❑ Si Netflix abona una cuota a Movistar, la red de contenidos de Netflix tendrá un mejor servicio a los usuarios de Movistar por lo que, si bien no hay acciones directas para priorizar su tráfico, el hecho es que tiene una ventaja sobre sus competidores.
- ❑ Si Netflix obtiene un acuerdo de “peering” por su celebridad, estaríamos igual que en el caso anterior, teniendo la misma ventaja sobre sus competidores.
- ❑ Si Netflix, u otro generador de contenidos, no tienen capacidad de tener red de contenidos propia, pero contrata a un proveedor existente, volvemos al primer caso.

Este escenario lleva a exponer que el mero uso de redes de contenidos, si bien no modifica las políticas de gestión de tráfico de los proveedores de servicios de internet, sí que obtiene ventanas en el tratamiento del tráfico y, por tanto, la red deja de ser neutra para cualquier generador de contenidos.

[16] https://ec.europa.eu/digital-agenda/en/net-neutrality-challenges
[17] https://www.netflix.com/es/

En resumen sobre este aspecto, no existe legislación al respecto actualmente, pero es posible que en un futuro cercano se establezca algún tipo de normativa para su utilización.

Por otro lado, es necesario tener cuenta la gestión de datos de carácter personal, que si tiene legislación aplicable en España, siendo esta la Ley Orgánica de Protección de Datos de Carácter Personal (15/1999), conocida como LOPD, siendo su última actualización de 2003.

En esta ley se establece en su artículo 3 que aquellos datos que “cualquier información concerniente a personas físicas identificadas o identificables.” es un dato de carácter personal. Es importante tener esto en cuenta debido a la distribución geográfica de los nodos de la red de contenidos, ya que esta ley, en título V, artículo 33 establece que:

1. No podrán realizarse transferencias temporales ni definitivas de datos de carácter personal que hayan sido objeto de tratamiento o hayan sido recogidos para someterlos a dicho tratamiento con destino a países que no proporcionen un nivel de protección equiparable al que presta la presente Ley, salvo que, además de haberse observado lo dispuesto en ésta, se obtenga autorización previa del Director de la Agencia de Protección de Datos, que sólo podrá otorgarla si se obtienen garantías adecuadas.

Por tanto, para respetar la legislación española, no es posible utilizar las redes de contenidos para la diseminación de bases de datos de usuarios para una aplicación o de los datos personales de cada uno.

Capítulo 8 - Conclusiones

Conclusiones

La utilización de infraestructuras de infraestructuras de tipo nube, si bien aumentan la complejidad de las infraestructuras de sistemas de información, también permiten una flexibilidad desconocidas hasta ahora. El hecho de poder crecer de forma horizontal en base a las necesidades de una organización sin tener que dimensionar al alza desde el comienzo ofrece una reducción sobre el coste total de propiedad muy alto.
Por otro lado, el hecho de quela plataformas de tipo nube ofrezcan servicios directamente a nivel de aplicación permite implementar aplicaciones de una forma muy rápida, centrándose en el desarrollo que aporta valor y dejando de lado muchos aspectos de infraestructura de los que la nube se encarga directamente. En este caso, el propio servicio de CDN se ha convertido en una pequeña aplicación de apenas una decena de líneas. Del mismo modo, eliminando del balanceador DNS todo lo relativo a sockets y tratamiento del protocolo DNS, también es extremadamente reducido, como lo es el notificador.

No obstante, el hecho más notable de utilizar una infraestructura es la posibilidad de dotarla de inteligencia propia mediante un orquestador que permite definir reglas para que la infraestructura se adecúe al entorno por sí misma de forma desatendida.

La implementación de esta CDN permite mostrar como puede ser sencillo montar este tipo de servicios si la infraestructura subyacente es la propicia.

Por último, con este entorno es posible demostrar como un servicio prestado desde un servidor tradicional puede delegar parte de su carga en este tipo de servicios.

Trabajo futuro

Con base en el trabajo realizado, sería posible mejorar sus prestaciones añadiéndole la siguiente funcionalidad:

Contabilidad de accesos por objeto a efectos de facturación por cliente y por zonas
El medio de negocio de las empresas de CDN es la facturación por su uso, es decir, por los usuarios a los que entregan objetos. Con la infraestructura mostrada se podría monitorizar el número de peticiones de objetos por contenedor, conociendo así lo que debería facturarse a cada cliente. Sin embargo, quizás es más caro prestar en una zona que en otra, por lo que además sería interesante monitorizar cuantos accesos se realizan desde cada zona. De este modo no sólo se podría facturar por accesos, si no también ponderando que no cuesta lo mismo establecer centros de datos en todos los sitios por igual.

API para almacenar objetos directamente
Sería posible realizar otra pequeña aplicación que ofreciese a los cliente una API que permitiese almacenar o eliminar objetos a cada cliente directamente mediante peticiones HTTP, pero cuya forma de acceso estuviese limitado por la base de datos de cliente y no se atacase directamente a la API de swift.

Interfaz de usuario de administración
El uso de la CDN debería ser desatendido por parte del prestador del servicio, por lo que sería interesante que existiese una aplicación web que permitiese a los clientes almacenar o eliminar objeto e, incluso, filtrar en que zonas deben servirse de una forma sencilla y amigable.

Servidor DNS balanceador geográfico
Una de las ventajas que ofrecen las CDN comerciales es que no sólo balancean entre los nodos que prestan servicio a una zona, si no que la propia resolución DNS ya lleva a la mejor zona para el usuario. Esto es, dada la dirección del usuario estima su ubicación y le redirije a la zona que puede prestarle un mejor servicio. Ya que existen bases de datos de direcciones IP geográficas, sería posible modificar el servidor DNS para que realice esta acción.

Escalar por zonas
El trabajo realizado sólo escala por tipo de nodo, no por zona, ya que se ha considerado que sólo existe una zona en todo el proyecto. Sería interesante modificar la plantilla del orquestador para que esta funcione aumentando los nodos en base a las zonas a las que presta servicio y no por el tipo de nodo.

Capítulo 9 - Presupuesto

Presupuesto

A continuación se detalla el presupuesto del proyecto para 48 jornadas, obtenido del diagrama de Gantt, con un jefe de proyecto dedicado a un 10% y un ingeniero a tiempo completo. También se incluye el coste del servidor donde se ha implementado.
No es necesario añadir ningún otro coste, ya todo el software utilizado es software libre.

En la siguiente página se incluye el presupuesto detallado.

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor: Fernando Cerezal López

2.- Departamento: Ingeniería Telemática

3.- Descripción del Proyecto:

- Título: **Montaje de una CDN sobre Openstack**
- Duración (meses): **2,2**

Tasa de costes Indirectos: 20%

4.- Presupuesto total del Proyecto (valores en Euros):

8.213,00 Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a título informativo)	Categoría	Dedicación (hombres mes) [a]	Coste hombre mes	Coste (Euro)	Firma de conformidad
					0,00	
Jaime García Reinoso		Ingeniero Senior	0,2	4.289,54	857,91	
Fernando Cerezal López		Ingeniero	2,2	2.694,39	5.927,66	
					0,00	
					0,00	
		Hombres mes	**2,4**	**Total**	6.785,57	

[a] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
Equipo HP Pavilion 500	999,00	100	2	60	36,63
Portátil HP ac006ns	599,00	100	2	60	21,96
		100		60	0,00
		100		60	0,00
		100		60	0,00
					0,00
				Total	58,59

[d] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} \times C \times D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[e]

Descripción	Empresa	Costes imputable
	Total	0,00

[e] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,...

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	6.786
Amortización	59
Subcontratación de tareas	0
Costes de funcionamiento	0
Costes Indirectos	1.369
Total	8.213

Figura 22: Presupuesto

Referencias

- ❑ Diseño e Implementación de un laboratorio de Openstack-Jesús Sánchez Manzanero
- ❑ Manual de Instalación de Openstack Juno[18]
- ❑ Documentación biblioteca de DNS para Python: dnslib[19]
- ❑ Documentación biblioteca de AMQP para Python: pika[20]
- ❑ Ejemplo de servidor DNS en Python con dnslib - ususario andreif - github[21]
- ❑ Microframework Flask[22]
- ❑ Documentación de referencia de Python[23]
- ❑ Documentación sobre libvirt de RedHat[24]
- ❑ Documentación de RabbitMQ[25]
- ❑ Ejemplo de autoescalado con Heat - Christian Berendt[26]
- ❑ Manual de instalación de ceilometer - RDO[27]
- ❑ Manual de Jmeter[28]

[18] http://docs.openstack.org/juno/install-guide/install/apt/content/index.html
[19] https://pypi.python.org/pypi/dnslib
[20] https://pypi.python.org/pypi/pika
[21] https://gist.github.com/andreif/6069838
[22] http://flask.pocoo.org/
[23] https://docs.python.org/2/
[24] https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/5/html/Virtualization/chap-Virtualization-Managing_guests_with_virsh.html
[25] https://www.rabbitmq.com/
[26] http://superuser.openstack.org/articles/simple-auto-scaling-environment-with-heat
[27] https://www.rdoproject.org/install/ceilometerquickstart/
[28] http://jmeter.apache.org/usermanual/

Glosario

AMQP: Advanced Message Queuing Protocol, protocolo de comunicación mediante bus de mensajería.
API: Aplication Programming Interface, conjunto de llamadas recursos que una solución expone para que puedan invocarse.
Bridge: Permite unir conexiones de red a nivel de enlace.
Caché: Memoria de uso intermedio donde se almacenan los datos con los que se está trabajando.
CDN: Content Delivery Network, red de entrega de contenidos.
Ceilometer: Módulo de openstack para la monitorización de la plataforma.
Cinder: Módulo de openstack para la gestión de almacenamiento de bloques.
Cliente: En este documento, organización que almacena objetos en una CDN para que sean accesibles a sus usuarios.
Css: Cascade Style Sheet, formato que permite definir formas y colores en una página web.
DHCP: Dynamic Host Configuration Protocol, protocolo que permite configurar automáticamente parámetros de red en un sistema.
DNS: Domain Name System, protocolo para la resolución de nombres de dominio a direcciones IP
Endpoint: URL donde se puede encontrar un servicio.
Flask: Microframework para la creación de aplicaciones web escrito en python
Framework: Conjunto de bibliotecas, funciones y otros recursos que permite construir soluciones sobre él.
Glance: Módulo de openstack para la gestión de imágenes de máquinas virtuales.
Heat: Módulo de openstack para la orquestación de la plataforma.
Horizon: Módulo de openstack para la gestión web de la plataforma.
HTTP: HyperText Transfer Protocol, protocolo para el intercambio de sitio web y elementos asociados.
JRE: Java Runtime Environment, entorno necesario para la ejecución de aplicaciones Java.
Json: Formato de intercambio de datos
Keystone: Módulo de openstack para la gestión y registro de usuarios, permisos y servicios.
Libvirt: Solución para la gestión de máquinas y redes virtuales.
LVM: Logical Volume Manager, gestor de volúmenes lógicos de almacenamiento en Linux.
Mac: dirección física de un disposito de red.
Mongodb: Gestor de base de datos NOSQL que utiliza json para el intercambio de datos.
Netns: net namespace, espacio virtual para la creación de redes

Neutron: Módulo de openstack para gestión de redes.
Nfs: Network File System, solución que permite servir sistemas de archivos por red.
Nova: Módulo de openstack para gestión de máquinas virtuales.
Openstack: Solución para la gestión de entornos virtuales en nube.
Openvswitch: Solución para la creación de redes virtuales.[29]
Proveedor de servicios de Internet: Organización que permite a otros acceder a Internet mediante sus medios, siendo los operadores de telecomunicaciones el caso más común.
Proxy: Servidor intermedio que recibe peticiones y las redirije a quien realmente presta el servicio.
Punto neutro: Ubicación donde los distintos Proveedores de servicios de internet intercomunican sus redes.
RabbitMQ: Servidor de bus de mensajería.
Rsync: Herramienta para la sincronización de árboles de sistemas de archivos.
Runlevel: Nivel de ejecución del sistema, cada nivel tiene asignados unos servicios que inicia o para.
Scp: Secure CP, variante de ssh para la copia de archivos remota.
Ssh: Secure Shell, herramienta para acceder remotamente a un sistema.
Swift: Módulo de openstack para el almacenamiento de objetos
System V: Tipo de sistema Unix cuyo arranque de servicios se realiza mediante scripts.
Tenant: Agrupación de elementos en openstack, también denominado proyecto.
Usuario: En este documento, usuario que accede a recursos web mediante un navegador web.
Virsh: Herramienta de consola de libvirt para la gestión de máquinas y redes virtuales.
Volumen de almacenamiento: Entidad que permite almacenar datos en su interior con un formato específico, similar a una partición de disco pero modificable dinámicamente.
Wget: Herramienta de consola para realizar peticiones web.
Zona: En este documento, subdominio bajo el que opera una CDN y el conjunto de nodos asociados.

[29] http://openvswitch.org/

ANEXO I - Código de servicio CDN

Código de servicio del nodo de CDN “cdnservice.py”

```
#!/usr/bin/python
import os
import swiftclient
from flask import Flask, abort

user = 'swift'
key = 'swift'
tenant = "service"
auth= "http://150.200.190.100:5000/v2.0/"

from flask import Flask
app = Flask(__name__)

conn = swiftclient.Connection(
        authurl=auth,
        user=user,
        key=key,
        tenant_name=tenant,
        auth_version="2.0"
        )

@app.route('/<client>/<objectId>')
def get(objectId, client):
    try:
        obj_tuple = conn.get_object(client, objectId)
        return obj_tuple[1]
    except:
        return abort(404)

if __name__ == "__main__":
    pid = os.getpid()
    os.system('echo "%s" > /var/run/cdn.pid' % pid)
    app.run(host='0.0.0.0', port=80)
```

ANEXO II - Script de arranque Notificador DNS

Código de script de notificador de balanceador DNS, cdndnsnotifier.py

```
#!/usr/bin/env python
### BEGIN INIT INFO
# Provides: cdndns
# Required-Start: networking
# Required-Stop:
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Description: CDN DNS Notifier
### END INIT INFO
import sys
import pika
import netifaces

credentials = pika.PlainCredentials('cdndns', 'cdndns')
ip=netifaces.ifaddresses('eth0')[2][0]['addr']
connection = pika.BlockingConnection(pika.ConnectionParameters(
        host='supervisor'))
channel = connection.channel()
channel.queue_declare(queue='cdndns')
messagemq=""
messageprint=""
zone="zone1"

def exiterror():
    print("usage cdndnsnotifier.py start|stop")
    sys.exit(1)

if len(sys.argv)!=2:
    exiterror()
command=str(sys.argv[1])

if command=="start":
    messagemq="add "+zone+" "+ip
    messageprint=" Request to add "+ip+" to "+zone
elif command=="stop":
    messagemq="del "+zone+" "+ip
    messageprint=" Request to delete "+ip+" from "+zone
else:
    exiterror()

channel.basic_publish(exchange='',
                      routing_key='cdndns',
                      body=messagemq)
print("CDNDNS: "+messageprint)
connection.close()
sys.exit(0)
```

ANEXO III - Código balanceador DNS

código de balanceador DNS, dns.py (basado en el script del usuario andreif en github: https://gist.github.com/andreif/6069838)

```
#!/usr/bin/python
# coding=utf-8
import re
import datetime
import sys
import time
import threading
import traceback
import SocketServer
import pika
import os
from dnslib import *

class DomainName(str):
    def __getattr__(self, item):
        return DomainName(item + '.' + self)

D = DomainName('example.com.')
IP = '222.222.222.222'
TTL = 60 * 5
PORT = 53

Domain= DomainName('.cdn.org.')
zones={}

soa_record = SOA(
    mname=D.ns1,  # primary name server
    rname=D.andrei,  # email of the domain administrator
    times=(
        201307231,  # serial number
        60 * 60 * 1,  # refresh
        60 * 60 * 3,  # retry
        60 * 60 * 24,  # expire
        60 * 60 * 1,  # minimum
    )
)
ns_records = [NS(D.ns1), NS(D.ns2)]
records = {
    D: [A(IP), AAAA((0,) * 16), MX(D.mail), soa_record] + ns_records,
    D.ns1: [A(IP)],  # MX and NS records must never point to a CNAME alias
(RFC 2181 section 10.3)
    D.ns2: [A(IP)],
    D.mail: [A(IP)],
```

```
    D.andrei: [CNAME(D)],
}


def dns_response(data):
    request = DNSRecord.parse(data)

    reply = DNSRecord(DNSHeader(id=request.header.id, qr=1, aa=1, ra=1),
q=request.q)

    qname = request.q.qname
    qn = str(qname)

    qtype = request.q.qtype
    qt = qtype


    if qn == Domain or qn.endswith(Domain):
        for zone in zones:
            if zone == qn:
                zones[str(qname)][0]=(zones[str(qname)][0]+1)%
(len(zones[str(qname)])-1)
                reply.add_answer(RR(rname=qname, rtype=1, rclass=1,
ttl=TTL, rdata=A(zones[str(qname)][zones[str(qname)][0]+1])))

    print "---- Reply:\n", reply

    return reply.pack()


class BaseRequestHandler(SocketServer.BaseRequestHandler):

    def get_data(self):
        raise NotImplementedError

    def send_data(self, data):
        raise NotImplementedError

    def handle(self):
        now = datetime.datetime.utcnow().strftime('%Y-%m-%d %H:%M:%S.%f')
        print "\n\n%s request %s (%s %s):" % (self.__class__.__name__[:3],
now, self.client_address[0],
                                              self.client_address[1])
        try:
            data = self.get_data()
            print len(data), data.encode('hex')  #
repr(data).replace('\\x', '')[1:-1]
            self.send_data(dns_response(data))
        except Exception:
            traceback.print_exc(file=sys.stderr)
```

```
class TCPRequestHandler(BaseRequestHandler):

    def get_data(self):
        data = self.request.recv(8192).strip()
        sz = int(data[:2].encode('hex'), 16)
        if sz < len(data) - 2:
            raise Exception("Wrong size of TCP packet")
        elif sz > len(data) - 2:
            raise Exception("Too big TCP packet")
        return data[2:]

    def send_data(self, data):
        sz = hex(len(data))[2:].zfill(4).decode('hex')
        return self.request.sendall(sz + data)


class UDPRequestHandler(BaseRequestHandler):

    def get_data(self):
        return self.request[0].strip()

    def send_data(self, data):
        return self.request[1].sendto(data, self.client_address)


def callback(ch, method, properties, body):
    print(" [x] Received %r" % body)
    m= re.match('(.*) (.*) (.*)', body)
    command= m.group(1)
    zone= m.group(2)+Domain
    address= m.group(3)
    if command == 'add':
      if zone in zones:
            zones[zone].append(address)
            zones[zone]=list(set(zones[zone]))
        else:
            zones[zone]=[0,address]#[iterator, node]

    if command == 'del':
        if zone in zones:
            if zones[zone].count(address) !=0:
                zones[zone].remove(address)
            if len(zones[zone])==1: #if only there is the counter
               del zones[zone]
    print zones

def worker():
    credentials = pika.PlainCredentials('cdndns', 'cdndns')
    connection = pika.BlockingConnection(pika.ConnectionParameters(
            host='supervisor'))
```

```
    channel = connection.channel()

    channel.queue_declare(queue='cdndns')


    channel.basic_consume(callback,
                          queue='cdndns',
                          no_ack=True)

    print(' [*] Waiting for messages. To exit press CTRL+C')
    channel.start_consuming()

   # return

if __name__ == '__main__':
    print "Starting nameserver..."
    pid = os.getpid()
    os.system('echo "%s" > /var/run/cdndns.pid' % pid)

    servers = [
        SocketServer.ThreadingUDPServer(('', PORT), UDPRequestHandler),
        SocketServer.ThreadingTCPServer(('', PORT), TCPRequestHandler),
    ]
    for s in servers:
        thread = threading.Thread(target=s.serve_forever)  # that thread
will start one more thread for each request
        thread.daemon = True  # exit the server thread when the main thread
terminates
        thread.start()
        print "%s server loop running in thread: %s" %
(s.RequestHandlerClass.__name__[:3], thread.name)
    listener = threading.Thread(target=worker)
    listener.start()

    try:
        while 1:
            time.sleep(1)
            sys.stderr.flush()
            sys.stdout.flush()

    except KeyboardInterrupt:
        pass
    finally:
        for s in servers:
            s.shutdown()
```

ANEXO IV - Script de inicio del servicio CDN

Script de arranque de cdnservice “runcdn”

```
#!/bin/bash


### BEGIN INIT INFO
# Provides: cdn
# Required-Start: networking
# Required-Stop:
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Description: CDN service
### END INIT INFO


case "$1" in
    start)
         cdnservice.py &
         echo "running cdnservice"
         ;;
    stop)
         kill `cat /var/run/cdn.pid`
         echo "stopping cdnservice"
         ;;

esac

exit 0
```

ANEXO V - Script de arranque balanceador DNS

Script de arranque del balanceador dns “dynamicdns.py”

```
 #!/bin/bash

### BEGIN INIT INFO
# Provides: dynamicdns
# Required-Start: networking
# Required-Stop:
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Description: Dynamic DNS
### END INIT INFO

case "$1" in
    start)
         /usr/bin/dns.py
         echo "running dynamic dns"
         ;;
    stop)
         kill `cat /var/run/cdndns.pid`
         echo "stopping dynamic dns"
         ;;
esac

exit 0
```

ANEXO VI – Script para obtener la IP externa

Script getexternalIP

#!/bin/bash
neutron --os-username admin --os-password s3cr3t0 --os-tenant-name caostack --os-auth-url http://150.200.190.100:35357/v2.0 floatingip-list | grep `ifconfig eth0 | grep "inet addr"| sed 's/.*addr:\([^\s]*\) .*/\1/' | sed -e 's/.*|.*| \(.*\)|.*|/\1/'